



EXASCALE
COMPUTING
PROJECT

ECP-U-ST-RPT_2022_00285

ECP Software Technology Capability Assessment Report V3.0

Michael A. Heroux, Director ECP ST
Lois Curfman McInnes, Deputy Director ECP ST
Rajeev Thakur, Programming Models & Runtimes Lead
Jeffrey S. Vetter, Development Tools Lead
Sherry Li, Mathematical Libraries Lead
James Ahrens, Data & Visualization Lead
Todd Munson, Software Ecosystem & Delivery Lead
Kathryn Mohror, NNSA ST Lead
Terece L. Turton, Integration Lead

June 1, 2022

ORNL/TM-2022/2651



U.S. DEPARTMENT OF
ENERGY

Office of
Science



National Nuclear Security Administration

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via US Department of Energy (DOE) SciTech Connect.

Website <http://www.osti.gov/scitech/>

Reports produced before January 1, 1996, may be purchased by members of the public from the following source:

National Technical Information Service

5285 Port Royal Road

Springfield, VA 22161

Telephone 703-605-6000 (1-800-553-6847)

TDD 703-487-4639

Fax 703-605-6900

E-mail info@ntis.gov

Website <http://www.ntis.gov/help/ordermethods.aspx>

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange representatives, and International Nuclear Information System representatives from the following source:

Office of Scientific and Technical Information

PO Box 62

Oak Ridge, TN 37831

Telephone 865-576-8401

Fax 865-576-5728

E-mail reports@osti.gov

Website <http://www.osti.gov/contact.html>

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ECP-U-ST-RPT_2022_00285

ECP Software Technology Capability Assessment Report V3.0

Office of Advanced Scientific Computing Research
Office of Science
US Department of Energy

Office of Advanced Simulation and Computing
National Nuclear Security Administration
US Department of Energy

June 1, 2022

ORNL/TM-2022/2651

REVISION LOG

Version	Date	Description
1.0	July 1, 2018	<i>ECP ST Capability Assessment Report</i>
1.5	February 1, 2019	<i>Second release</i>
2.0	February 1, 2020	<i>Third release</i>
2.5	November 19, 2020	<i>Fourth release</i>
3.0	November 22, 2021	<i>Fifth release-DRAFT</i>
3.0	March 1, 2022	<i>Fifth release-DRAFT V2</i>
3.0	June 1, 2022	<i>Fifth release</i>

EXECUTIVE SUMMARY

The Exascale Computing Project (ECP) Software Technology (ST) focus area is responsible for (1) developing critical software capabilities that will enable the successful execution of ECP applications and (2) providing key components of a productive and sustainable exascale computing ecosystem that will position the US Department of Energy (DOE) and the broader high-performance computing (HPC) community with a firm foundation for future extreme-scale computing capabilities.

This ECP ST Capability Assessment Report (CAR) provides an overview and assessment of current ECP ST capabilities and activities, giving stakeholders and the broader HPC community information that can be used to assess ECP ST progress and plan their own efforts accordingly. ECP ST leaders commit to updating this document on regular basis (every 6–12 months). Highlights from this version of the report are presented here.

This version of the CAR contains the following updates relative to the previous revision.

- This report highlights the progress with the Extreme-scale Scientific Software Stack (E4S) efforts. In particular, this report discusses how E4S continues to gain traction as a first-class entity in the HPC ecosystem, enabling new conversations with users, facilities, vendors, other US agencies, and international partners.
- The several-page summaries of each ECP Level 4 project were updated to reflect recent progress and next steps (Section 4). Of particular note are the experiences of our teams on early-access systems for Frontier.
- The E4S is described further. E4S is now updated via quarterly releases. E4S is the primary integration and delivery vehicle for ECP ST capabilities (Section 2.1.1).
- The ECP ST software development kit (SDK) effort further refined its groupings (Section 2.1.2).

The ECP ST focus area represents the key bridge between exascale systems and the scientists developing applications that will run on those platforms. ECP ST efforts contribute to approximately 70 software products (Section 2.1.3) in six technical areas (Table 1). Since publishing the previous revision of the CAR, the team has continued to evolve the product dictionary of official product names, which enables more rigorous mapping of ECP ST deliverables to stakeholders (Section 2.1.4).

Programming Models & Runtimes: In addition to developing key enhancements to MPI and OpenMP for scalable systems with accelerated node architectures, the team is working on performance portability layers (Kokkos and RAJA) and participating in OpenMP and OpenACC software design and development that will enable applications to write much of their source code without needing to provide vendor-specific implementations for each exascale system. One legacy of ECP ST efforts is anticipated to be a software stack that supports Intel and AMD accelerators in addition to NVIDIA’s accelerators (Section 4.1).

Development Tools: The team is enhancing existing widely used compilers (e.g., LLVM) and performance tools for next-generation platforms. Compilers are critical for heterogeneous architectures, and LLVM is the most popular compiler for heterogeneous systems. As node architectures become more complicated and concurrency becomes even more necessary, compilers must generate optimized code for many architectures, and the impediments to performance and scalability will become even more difficult to diagnose and fix. Development tools provide essential insight into these performance challenges and code transformation and support capabilities that help software teams generate efficient code, use new memory systems, and more (Section 4.2).

Mathematical Libraries: High-performance scalable math libraries have enabled parallel execution of many applications for decades. The ECP ST is providing the next generation of these libraries to address needs for latency hiding, improved vectorization, threading, and strong scaling. Additionally, the team is addressing new demands for system-wide scalability, including improved support for coupled systems and ensemble calculations (Section 4.3). The Mathematical Libraries teams are also leading the SDK initiative that is a pillar of the ECP ST software delivery strategy (Section 2.1.2). Recently, the FFTX portion of the STRUMPACK/SuperLU project was transferred out of ECP ST into the ECP AD co-design portfolio, to better align the exploratory nature of FFTX with co-design approaches.

Data & Visualization: The ECP ST has a large collection of data management and visualization products that provide essential capabilities for compressing, analyzing, moving, and managing data. These tools are becoming even more important as the volume of simulation data produced grows faster than the ability to capture and interpret it (Section 4.4).

Software Ecosystem & Delivery: This technical area provides important enabling technologies, such as Spack [1], a from-source build and package manager; container environments for high-performance computers; and a toolkit of reusable components for scientific workflow management systems. This area also provides the critical resources and staffing that will enable the ECP ST to perform continuous integration testing and product releases. Finally, this area engages with software and system vendors and DOE facilities staff to ensure the coordinated planning and support of ECP ST products (Section 4.5).

NNSA ST: This technical area brings all the National Nuclear Security Administration (NNSA)-funded work in the ECP ST into one Level 3 (L3) area for easier coordination with other project work at NNSA laboratories. Introducing this L3 area enables continued integrated planning with the rest of the ECP ST while permitting flexible coordination within the NNSA laboratories (Section 4.6).

ECP ST Software Delivery Mechanisms: The ECP ST delivers software capabilities to users via several mechanisms (Section 3). Almost all products are delivered via source code to at least some of their users. About 10 products are available via vendor software stack and via binary distributions, such as Linux distributions.

ECP ST Project Overviews: A significant portion of this report includes two-page synopses of each ECP ST project (Section 4), including a project overview, key challenges, solution strategy, recent progress, and next steps.

Project Organization: The ECP ST has established a tailored project management structure using capability integration goals, milestones, regular project-wide video meetings, monthly and quarterly reporting, and an annual review process. This structure supports project-wide communication and coordinated planning and development that enables 35 projects and more than 250 contributors to create the ECP ST software stack.

How to navigate this document: This report is large. However, it is organized in a hierarchical fashion that should permit rapid navigation for readers who are interested in specific information. The report is organized as follows.

- General overview of ECP ST: Sections 1–3 provide an introduction and general overview of the ECP ST.
- Section 4 is dedicated to the L3 technical areas. Each subsection includes an overview of the L3 area followed by a two-page overview, status, and plans for each product supported by the ECP ST.
 - Programming Models & Runtimes: Section 4.1.
 - Development Tools: Section 4.2.
 - Mathematical Libraries: Section 4.3.
 - Data & Visualization: Section 4.4.
 - Software Ecosystem & Delivery: Section 4.5.
 - NNSA ST: Section 4.6.

TABLE OF CONTENTS

EXECUTIVE SUMMARY	v
LIST OF FIGURES	x
LIST OF TABLES	xvi
1 Introduction	1
1.1 Background	2
1.2 ECP ST Project WBS Changes	4
2 ECP ST Planning, Execution, Tracking and Assessment	10
2.1 ECP ST Architecture and Design	10
2.1.1 The Extreme-Scale Scientific Software Stack	10
2.1.2 Software Development Kits	15
2.1.3 ECP ST Product Dictionary	16
2.1.4 ECP Product Dependency Management	17
2.2 ECP ST Planning and Tracking	20
2.2.1 ECP ST P6 Activity Issues	20
2.2.2 KPP-3	20
2.2.3 ECP ST Software Delivery	23
2.2.4 ECP ST Software Life Cycle	25
3 ECP ST Deliverables	27
3.1 ECP ST Development Projects	27
3.2 Standards Committees	29
3.3 Contributions to External Software Products	30
4 ECP ST Technical Areas	32
4.1 WBS 2.3.1 Programming Models & Runtimes	32
4.1.1 Scope and Requirements	32
4.1.2 Assumptions and Feasibility	33
4.1.3 Objectives	33
4.1.4 Plan	33
4.1.5 Risks and Mitigation Strategies	34
4.1.6 Future Trends	34
4.1.7 WBS 2.3.1.01 Programming Models & Runtimes Software Development Kits	35
4.1.8 WBS 2.3.1.07 Exascale MPI	35
4.1.9 WBS 2.3.1.08 Legion	37
4.1.10 WBS 2.3.1.09 Distributed Tasking at Exascale: PaRSEC	38
4.1.11 WBS 2.3.1.14 GASNet-EX	44
4.1.12 WBS 2.3.1.14 UPC++	46
4.1.13 WBS 2.3.1.16 SICM	49
4.1.14 WBS 2.3.1.17 Open MPI for Exascale (OMPI-X)	50
4.1.15 WBS 2.3.1.18 RAJA/Kokkos	54
4.1.16 WBS 2.3.1.19 Argo: Low-Level Resource Management for the OS and Runtime	57
4.2 WBS 2.3.2 Development Tools	66
4.2.1 Scope and Requirements	66
4.2.2 Assumptions and Feasibility	67
4.2.3 Objectives	67
4.2.4 Plan	67
4.2.5 Risk and Mitigation Strategies	67
4.2.6 Future Trends	68

4.2.7	WBS 2.3.2.01 Development Tools Software Development Kits	68
4.2.8	WBS 2.3.2.06 Exa-PAPI	70
4.2.9	WBS 2.3.2.08 HPCToolkit	72
4.2.10	WBS 2.3.2.10 PROTEAS-TUNE: Programming Toolchain for Emerging Architectures and Systems	74
4.2.11	WBS 2.3.2.10 PROTEAS-TUNE: LLVM	76
4.2.12	WBS 2.3.2.10 PROTEAS-TUNE - Clacc: OpenACC in Clang and LLVM	78
4.2.13	WBS 2.3.2.10 PROTEAS-TUNE - LLVM-DOE: Creating and Maintaining a DOE Fork of LLVM	80
4.2.14	WBS 2.3.2.10 PROTEAS-TUNE - FLACC and MLIR: Creating and Maintaining OpenACC in LLVM/Flang	81
4.2.15	WBS 2.3.2.10 PROTEAS-TUNE: Autotuning	82
4.2.16	WBS 2.3.2.10 PROTEAS-TUNE - Bricks	85
4.2.17	WBS 2.3.2.10 PROTEAS-TUNE - TAU Performance System	86
4.2.18	WBS 2.3.2.10 PROTEAS-TUNE - PAPHYRUS: Parallel Aggregate Persistent Storage	89
4.2.19	WBS 2.3.2.10 PROTEAS-TUNE - SYCL	90
4.2.20	WBS 2.3.2.11 SOLLVE	91
4.2.21	WBS 2.3.2.11 Argobots: Flexible, High-Performance Lightweight Threading	93
4.2.22	WBS 2.3.2.11 BOLT: Lightning Fast OpenMP	95
4.2.23	WBS 2.3.2.11 LLVM Implementations	97
4.2.24	WBS 2.3.2.11 Building tools to detect and debug bugs and performance issues with OMP offloading	103
4.2.25	WBS 2.3.2.11 Validation and Verification Testsuite	104
4.2.26	WBS 2.3.2.11 Training & Outreach in 2021	105
4.2.27	WBS 2.3.2.12 Flang	106
4.3	WBS 2.3.3 Mathematical Libraries	107
4.3.1	Scope and Requirements	107
4.3.2	Assumptions and Feasibility	108
4.3.3	Objectives	108
4.3.4	Plan	108
4.3.5	Risks and Mitigation Strategies	109
4.3.6	Future Trends	109
4.3.7	WBS 2.3.3.01 xSDK	110
4.3.8	WBS 2.3.3.01 xSDK Sub-project: multiprecision	112
4.3.9	WBS 2.3.3.01 xSDK Sub-project: batched sparse linear algebra	114
4.3.10	WBS 2.3.3.06 PETSc-TAO	115
4.3.11	WBS 2.3.3.07 STRUMPACK-SuperLU	117
4.3.12	WBS 2.3.3.12 Sub-project: SUNDIALS	119
4.3.13	WBS 2.3.3.12 Sub-project: hypre	122
4.3.14	WBS 2.3.3.13 CLOVER	125
4.3.15	WBS 2.3.3.13 CLOVER Sub-project: heFFTe	126
4.3.16	WBS 2.3.3.13 CLOVER Sub-project: SLATE	128
4.3.17	WBS 2.3.3.13 CLOVER Sub-project: Ginkgo	131
4.3.18	WBS 2.3.3.14 ALExa	133
4.3.19	WBS 2.3.3.15 Sake	137
4.3.20	WBS 2.3.3.15 Sake Sub-project: Kokkos Kernels	137
4.3.21	WBS 2.3.3.15 Sake Sub-project: Trilinos/PEEKS	139
4.4	WBS 2.3.4 Data & Visualization	140
4.4.1	Scope and Requirements	141
4.4.2	Assumptions and Feasibility	141
4.4.3	Objectives	142
4.4.4	Plan	143
4.4.5	Risks and Mitigation Strategies	143
4.4.6	Future Trends	144

4.4.7	WBS 2.3.4.01 Data & Visualization Software Development Kits	147
4.4.8	WBS 2.3.4.09 ADIOS	150
4.4.9	WBS 2.3.4.10 DataLib	152
4.4.10	WBS 2.3.4.13 ECP/VTK-m	155
4.4.11	WBS 2.3.4.14 VeloC: Very Low Overhead Checkpointing System	158
4.4.12	WBS 2.3.4.14 ECP SZ: Fast, Effective, Parallel Error-bounded Exascale Lossy Compression for Scientific Data	160
4.4.13	WBS 2.3.4.15 ExaIO - ExaHDF5	163
4.4.14	WBS 2.3.4.15 UnifyFS – A File System for Burst Buffers	165
4.4.15	WBS 2.3.4.16 ALPINE	167
4.4.16	WBS 2.3.4.16 ZFP: Compressed Floating-Point Arrays	170
4.5	WBS 2.3.5 Software Ecosystem & Delivery	172
4.5.1	Scope and Requirements	172
4.5.2	Assumptions and Feasibility	173
4.5.3	Objectives	173
4.5.4	Plan	173
4.5.5	Risks and Mitigation Strategies	174
4.5.6	Future Trends	174
4.5.7	WBS 2.3.5.01 Software Development Kits	175
4.5.8	WBS 2.3.5.09 Software Packaging Technologies	178
4.5.9	WBS 2.3.5.10 ExaWorks	180
4.6	WBS 2.3.6 NNSA ST	182
4.6.1	Scope and Requirements	182
4.6.2	Objectives	182
4.6.3	Plan	182
4.6.4	Risks and Mitigation Strategies	182
4.6.5	WBS 2.3.6.01 LANL ATDM Software Technologies	183
4.6.6	WBS 2.3.6.02 LLNL ATDM Software Technologies	188
4.6.7	WBS 2.3.6.03 Sandia ATDM Software Technologies	195
5	Conclusion	201

LIST OF FIGURES

1	The ECP ST before the November 2017 reorganization. This conceptual layout emerged from several years of exascale planning conducted primarily within DOE ASCR. Because of significant restructuring of the ECP that removed many of the facility’s activities and reduced the project time line from 10 to 7 years, as well as a growing awareness of what risks had diminished, this diagram no longer represented the ECP ST efforts accurately.	4
2	The ECP ST after the November 2017 reorganization. This diagram more accurately reflects the priorities and efforts of the ECP ST given the new ECP project scope and the demands foreseen.	5
3	The ECP ST after the October 2019 reorganization. This diagram reflects the further consolidation of NNSA open-source contributions to enable the more flexible management of NNSA ST contributions.	5
4	Project remapping summary from Phase 1 (through November 2017) to Phase 2 (November 2017–September 30, 2019) to Phase 3 (After October 1, 2019).	6
5	The ECP WBS through Level 3 (L3) as of August 2022. Under ST, WBS 2.3.6 consolidates ATDM contributions to the ECP into a new L3 area.	7
6	The FY22 ECP ST WBS as of June 1, 2022. Several teams have had PI changes since the last version of this chart. We have also added a new role of integration lead, led by Terry Turton, to work with coordination of capability integrations on our way to completing KPP-3.	8
7	The ECP ST Leadership Team as of November 2020. Jonathan Carter, previous deputy director of the ECP ST, became associate laboratory director of the Computing Sciences Area at Lawrence Berkeley National Laboratory. His departure led to Lois Curfman McInnes, previously the L3 lead of Mathematical Libraries, being named the ECP ST deputy director and Sherry Li being named the new Mathematical Libraries L3 lead. Rob Neely was also promoted at Lawrence Livermore National Laboratory (LLNL), leading to Kathryn Mohror becoming the L3 lead of NNSA ST.	9
8	Using Spack [2], E4S builds a comprehensive software stack. As ECP ST efforts proceed, E4S will be used for continuous integration testing, providing developers with rapid feedback on regression errors and providing user facilities with a stable software base as the team prepares for exascale platforms. This diagram shows how E4S builds ECP products via an SDK target, which are the math libraries’ SDK, called xSDK in this example. The SDK target then builds all products that are part of the SDK (see Figure 12 for SDK groupings), first defining and building external software products. Green-labeled products are part of the SDK. The blue label indicates expected system tools, which, in this case, is a particular version of Python. Black-labeled products are expected to be previously installed into the environment, which is a common and easily satisfied requirement. Using this approach, users interested in only SUNDIALS, a particular math library, can be assured that the SUNDIALS build will be possible because it is a portion of what E4S builds and tests.	11
9	Using Spack build cache features, E4S builds can be accelerated via cached binaries for any build signature that Spack has already seen. Between September 2020 and June 2022, more than 42,000 binaries were added to the cache.	13
10	E4S now supports Google Cloud Platform in addition to Amazon Web Services (AWS).	14
11	Version 1 of the E4S community policies. These policies serve as membership criteria for E4S member packages. The E4S community policy effort heavily leveraged the existing xSDK community policies [3].	15
12	The breakdown of ECP ST products into six SDKs are shown in the first six columns. The rightmost column lists products that are not part of an SDK but are part of the Ecosystem group that will also be delivered as part of the E4S. Section 4.5.7 provides an update on the progress in defining SDK groupings.	17
13	Screenshot from the top of the ECP Confluence wiki page containing the <i>ECP ST Product Dictionary</i> . The product dictionary structure contains primary and secondary products. Client (i.e., consumer) dependencies are stated against the primary product names only, enabling unambiguous mapping of AD-on-ST and ST-on-ST dependencies.	18

14	Screenshots of the <i>ECP ST Product Dictionary</i> table. The table is actively managed to include primary and secondary products to which the ECP ST team contributes and upon which ECP ST clients depend. Presently, the product dictionary contains 70 primary products. Secondary products are listed under the primary product with the primary product as a prefix. For example, AID is the second primary product in the table shown here. STAT, Archer, and FLIT are component subproducts. MPI (not shown) is another primary product. MPICH and OpenMPI are two robust MPI implementations and are listed as MPI subproducts.	18
15	Using Jira, the ECP manages its AD, ST, HI, vendor, and facilities dependencies. This figure shows a screenshot of the Jira dependency database dashboard and an edit panel, which supports the creation and management of a consumer-on-producer dependency.	19
16	This query result from the ECP Jira dependency database lists all consumers of capabilities from the PETSc/TAO product. Selecting the details of one of the dependency issues shows how critical the dependency is and any custom information peculiar to the particular dependency.	19
17	Example of a P6 Activity.	20
18	The ECP ST software stack is delivered to the user community through several channels, including via source code, using SDKs, direct to facilities in collaboration with ECP HI, and via binary distributions from containers and HPC vendors. Increasingly, E4S is the primary pathway for delivering ECP ST capabilities. E4S provides testing, a documentation portal, and quality commitments via community policies.	24
19	ECP ST product planning, execution, testing, and assessment are governed by the combination of P6 Activities for hierarchical planning (Figure 17) and KPP-3 for measuring capability integrations (Table 4). This figure shows how the entire life cycle of ECP ST feature development is captured by these two elements.	26
20	ECP ST is composed of six L3 technical areas. The first four areas are organized around functionality development themes. The fifth is focused on technology for packaging and delivering capabilities. The sixth is organized around per-lab open-source development at the three DOE NNSA laboratories: LANL, LLNL, and Sandia.	32
21	ParSEC modular framework allows each component to be dynamically activated.	39
22	Time to solution and Performance as a function of the number of V100 GPUs on Summit, for the molecule $C_{65}H_{132}$	40
23	Comparison of DPLASMA and SLATE Cholesky factorization over ParSEC with SLATE and ScaLAPACK on 64 nodes 12 cores each	40
24	Performance using GPUs of native ScaLAPACK, ScaLAPACK over DPLASMA and native DPLASMA for GEMM and POTRF.	41
25	Performance comparison between DPLASMA (PTG DSL), the TTG implementation, and ScaLAPACK, of the Cholesky factorization (double precision), for a fixed amount of memory per node (weak scaling).	42
26	Performance of ParSEC on AMD accelerated system OLCF Spock.	43
27	Selected GASNet-EX vs. MPI RMA Performance Results	45
28	Mean run time per step (lower is better) in a strong scaling study comparing UPC++ (solid series) and MPI versions (dashed series) of a GPU-enabled 3D heat conduction example code, run on OLCF's Summit for three problem sizes (given as length of one edge of the cubic domain).	47
29	Preliminary flood bandwidth on OLCF's Spock of RMA put operations from local host memory to remote GPU memory, implemented with and without use of the ROCmRDMA capability in GASNet-EX, along with the bandwidth ratio.	48
30	Data tiering with online application guidance. (a) Users first compile the application with a custom pass to insert annotations at each allocation call site, (b) Program execution proceeds inside the SICM runtime layer, which automatically profiles memory usage behavior, converts it into tier recommendations for each allocation site, and enforces these recommendations during program execution. In (b), interactions and operations drawn with dashed lines only occur at regular, timer-based intervals, while the solid lines correspond to activities that can occur throughout the program execution [4].	51
31	Schematic code illustrating use of partitioned communications from a GPU kernel. Host code (left) sets up communication, GPU kernel (right) triggers send (MPI_Pready).	52

32	Comparison of recovery times for traditional checkpoint/restart (blue) and Reinit (green) for the HPCG benchmark. Each graph represents different types/severities of failures.	53
33	Illustration of relative costs of PThreads vs ULT threads (Qthreads, Argobots) using ThreadOps benchmark for fork-join and yield operations.	53
34	AML components.	58
35	Performance of AML replicaset on different hardware architectures	59
36	UMap Handler architecture	61
37	Envisioned PowerStack	62
38	NRM sensor output.	65
39	Control Problem Description (CPD).	65
40	Existing PAPI SDE C API and new SDE C++ API.	70
41	(a) HPCToolkit's <code>hpcviewer</code> showing a detailed attribution of GPU performance metrics in a profile of PeleC. (b) HPCToolkit's <code>hpctraceviewer</code> showing CPU and GPU trace lines for LAMMPS.	73
42	Clacc workflow.	79
43	Bricks can be used to map memory onto regions that eliminate ghost zone packing and MPI message count (2D example).	86
44	TAU was used to collect profiles and traces of ECP proxy applications like miniFE (trace shown in Vampir), observing OpenMP parallel regions, loops and synchronization without application instrumentation.	87
45	TAU was used to collect profiles and traces of OpenACC benchmarks (303.stencil trace shown in Vampir), observing OpenACC regions and device offload events without application instrumentation.	88
46	SOLLVE thrust area updates.	93
47	Argobots execution model.	94
48	MPI+Threads interoperability of BOLT. OpenMP threads and tasks in BOLT interact MPI implementations via the Argobots layer.	96
49	Conceptual lowering of the <code>target</code> directive to the hidden helper task design. A special <code>hht_task</code> is used and executed by an hidden helper thread while the offload part is made synchronous.	98
51	Performance impact of loop transformations on CPUs.	99
53	Performance results for the OpenMC [5] OpenMP offloading code that simulates transport of neutral particles using the Monte Carlo methodology. The application is part of the ExaSMR ECP project and known for its proxy applications (XSBench [6] and RSBench [7]). Results show the almost 100x speedup achieved by co-optimizing the full OpenMC application with the LLVM compiler for OpenMP offloading on NVIDIA A100 GPUs. In the process, changes to both OpenMC and LLVM were made, and the latter were often triggered via command line options or assumptions. Guided by compiler remarks, other applications could benefit similarly and with far less involvement from a compiler developer—but only if the remarks, and later the suggested code changes, are clear, actionable, and effective. For context, we also present the AMD MI100 numbers obtained with the latest versions of OpenMC and the LLVM/Clang compiler. Since the LLVM/OpenMP offloading backend for AMD GPU is new and only recently became stable, we expect substantial performance improvements as we start our tuning efforts. Even without, we believe the results shows how performance portability is certainly within reach for a full application using the LLVM/OpenMP offloading environment. The data for the figure was generated and generously provided by John Tramm.	101
54	Running AutoDock-GPU with LLVM's OpenMP on Spock.	102
55	Dependency graph of xSDK packages with Spack-enabled interoperabilities represented in xSDK 0.6.0. A→B indicates A uses B.	111
56	Evolution of the machine balance of processors over different hardware generations.	113
57	The PETSc/TAO architecture enables users to utilize a variety of programming models for GPUs independently of PETSc's internal programming model.	116

58	Solve time for a 3D Laplacian with second-order elements. Larger grids are generated by uniform refinement. Runs are configured with six resource sets on each Summit node, each with one GPU and 4 MPI processes.	116
59	STRUMPACK numerical factorization performance on Summit.	119
60	Runtimes from a pre-mixed flame test case in PeleC using various time integrators from SUNDIALS for chemistry evolution. Demonstrated 44% reduction in runtime through algorithm choice using 2 nodes and 12 GPUs on Summit. Note that more efficient algorithms required fewer calls to the expensive right-hand-side function.	121
61	Speedup of 2D diffusion on one Spock node with differing problem sizes. CPU problems used 60 CPU cores and GPU problems used all 4 MI100 GPUs on the node.	122
62	Speedup of 2D diffusion with 10^7 unknowns on 1 MPI task with 1 MI100 GPU on Spock and 1 GCD of a MI250X GPU on Crusher.	123
63	Solution of a simple 3D electromagnetic diffusion problem corresponding to the 2nd order definite Maxwell equation $\nabla \times \nabla \times E + E = f$ using AMS-PCG on 1 node of Lassen (4 GPUs vs. 40 CPU cores) using finite elements of increasing order on a Fichera mesh.	124
64	Total times (setup plus solve times) on 2 nodes of Spock using AMG-PCG with aggressive coarsening and multi-pass interpolation for a 3D diffusion problem with a 27-point stencil on a $n \times n \times n$ grid. The CPU runs were performed with 8 MPI tasks with 16 OpenMP threads per MPI task, the GPU runs were performed on 8 AMD MI100 GPUs.	124
65	Setup, solve times and Speedups of GPU over CPU performance on 1 node of Crusher using AMG-PCG with three different settings for a 3D diffusion problem with a 27-point stencil on a $n \times n \times n$ grid. The CPU runs were performed with 64 MPI tasks, the GPU runs were performed on 8 AMD MI250 GPUs. Solid lines use the GPU default settings, whereas dashed and dotted lines use in addition one level of aggressive coarsening with two-stage and multipass interpolation, respectively.	125
66	Left: heFFTe software stack. Right: 3D FFT computational pipeline in heFFTe with (1) flexible API for application-specific input and output, including bricks/pencils; (2) efficient packing/unpacking and MPI communication routines; and (3) efficient 1D/2D/3D FFTs on the node.	126
67	Left: heFFTe strong scalability on 1024^3 FFT on up to 256 nodes ($6 \times V100$ GPUs; double complex arithmetic; starting and ending with bricks; performance assumes $5N^3 \log_2 N^3$ flops). Middle: Convolution of 512^3 multidimensional arrays in double complex arithmetic. Right: Performance of the Discrete Sine Transformation (DST) in heFFTe 2.2.	127
68	Speedup of batching FFTs using heFFTe on Spock. Shown is strong scalability of batched vs. not batched FFTs of size 32^3 , which are realistic application sizes as needed in NWChemEx.	128
69	SLATE in the ECP software stack.	129
70	Performance improvement in Hermitian reduction to band for eigenvalue problem. Each node has 2×10 core Intel Haswell E5-2650 v3.	130
71	Performance on Spock (4 nodes, 16 AMD MI100 GPUs).	130
72	The portability design of the Ginkgo math library enables high performance kernel implementations in the vendor-supported programming language.	131
73	Achieved bandwidth (GB/s) of the Ginkgo CSR (left), and ELL (right) SpMV operations on one GCD of the Crusher MI250X GPUs.	133
74	Achieved Bandwidth (GB/s) of the Ginkgo CG (left) and GMRES (right) solvers on one GCD of the Crusher MI250X GPUs.	133
75	A notional diagram of DOE facility storage resources. Not all systems have each role filled, and often additional network connections exist to accelerate specific data flows.	145
76	HDF5 architecture: green rectangles represent dynamically loaded VFD and VOL connectors to access data on different storage devices; grey and blue rectangles represent components of the HDF5 library.	149
77	An example of using ADIOS to support ECP science. This sketch represents the demonstration at the February 2018 ECP Meeting, which featured WDM Fusion, CODAR, ADIOS, and other joint ECP activities. Note that all of the green arrows in the figure represent data communication or storage handled by the ADIOS infrastructure.	152

78	Examples of recent progress in VTK-m include (from left to right) in situ visualization with WDMApp simulations, particle density estimation, flow in laser wakefields, and in situ visualization with WarpX simulations.	156
79	Performance improvement of VTK-m’s unstructured gradient benchmark under Kokkos.	158
80	VeloC architecture.	159
81	Parallelism implemented for Huffman coding’s subprocedures (kernels): <i>sequential</i> denotes that only 1 thread is used due to data dependency, <i>coarse-grained</i> denotes that data is explicitly chunked, and <i>fine-grained</i> denotes that there is a data-thread mapping with little or no warp divergence.	162
82	An overview of asynchronous I/O as an HDF5 VOL connector.	164
83	Using UnifyFS from an MPI application is as easy as using the parallel file system. Simply change the file path to point to the UnifyFS mount point <code>/unifyfs</code> and then perform I/O as normal.	166
84	UnifyFS Overview. Users can give commands in their batch scripts to launch UnifyFS within their allocation. UnifyFS works transparently with POSIX I/O, common I/O libraries, and VeloC. Once file operations are intercepted by UnifyFS, they are handled with specialized optimizations to ensure high performance.	167
85	The ALPINE statistical feature detection algorithm is used to identify bubbles in situ in an MFiX-Exa fluidized bed simulation. The raw article data is converted to a particle density field. A threshold is applied to the density field to create a feature similarity field, separating the bubbles from uninteresting regions. Saving only the statistical representation allows greater temporal resolution while significantly reducing output data size. A preliminary study shows a factor of 300 reduction in data size compared to the raw particle fields. The statistical bubble representation becomes the input to a post hoc Cinema-based workflow to track bubbles and explore bubble dynamics.	170
86	ZFP (de)compression performance.	172
87	250:1 ZFP compression of EQSIM seismic wave velocity data (figure courtesy of McCallen et al. [8]).	172
88	WDMapp documentation for how to use the E4S WDMapp Docker container to speed up WDMapp installation by leveraging the E4S Spack build cache.	176
89	Output from the Nalu-Wind application built using E4S, is visualized using ParaVer and TAU on an AWS cloud instance.	177
90	Pantheon Science uses E4S build caches to speed up installation on Summit at ORNL.	178
91	ExaWorks tool kit.	180
92	Productivity features such as dynamic control replication scales well across multiGPU systems in unstructured mesh computations.	185
93	New Legion features such as tracing will improve strong scaling in unstructured mesh computations.	185
94	In this example of the new Cinema CIS format, a groundwater simulation can be split into water streamlines and stone layers. From left to right: the stone layer has a cool-warm colormap applied post hoc; the same stone layer in grayscale; the water streamlines are colormapped in inferno; the stone layer and the water streamline layers are separately colormapped and then composited to form the final visualization image.	187
95	In this example of a curated Pantheon workflow, the Nalu-Wind simulation outputs a data file which is input to a ParaView script that outputs a Cinema database of visualizations that then go through a validation step to verify the correctness of the Cinema database.	187
96	AMR implementation in MFEM allows many applications to benefit from non-conforming adaptivity, without significant changes in their codes.	190

97 An illustration of how the computing resources allocated to a job—as granted by the HPC workload manager—differs between conventional and emerging scientific workflows. The notional X-axes depict the compute node IDs allocated to the job and Y-axes depict the IDs of computing resources in each of these nodes. (a) The conventional paradigm requires only a single parallel simulation application to run; (b) The emerging paradigm often requires many different types of tasks such as an ensemble of molecular dynamic (MD) parallel simulation applications and another ensemble of docking simulation applications along with in situ data analysis for the MD ensemble while these tasks are driven by an AI. 193

98 Spack build pipelines at facilities will provide HPC-native binary builds for users. 194

99 The MFEM team added support for variable order FEM spaces with general hp-refinement so higher order apps can better adapt to solutions. 194

100 Kokkos Execution and Memory Abstractions 197

LIST OF TABLES

1	ECP ST Work Breakdown Structure (WBS), technical area, and description of scope.	2
2	E4S provides a complete Linux-based software stack that is suitable for many scientific workloads, tutorial, and development environments. Additionally, it is an open-software architecture that can expand to include any additional and compatible Spack-enabled software capabilities. Because Spack packages are available for many products and easily created for others, E4S is practically expandable to include almost any robust Linux-based product. Furthermore, E4S capabilities are available as subtrees of the full build. E4S is not monolithic.	14
3	The ECP’s four KPPs.	21
4	Integration goal scoring. One point is accrued when a client integrates and sustainably uses a product’s capabilities. Scores are assessed annually.	22
5	Key metric values. These values are determined by the L4 subproject team when defining its KPP-3 issue.	22
6	Integration scores. Each integration score will have an associated weight, depending on the potential impact if integration targets are not met.	23
7	Programming Models & Runtimes (18 total).	27
8	Development Tools (22 total).	28
9	Mathematical Libraries (18 total).	28
10	Data & Visualization (25 total).	29
11	Software Ecosystem & Delivery (two total).	29
12	ECP ST staff are involved in a variety of official and de facto standards committees. Involvement in standards efforts is essential for ensuring the sustainability of ECT ST products and ensuring that emerging exascale requirements are addressed by these standards.	30
13	External products to which ECP ST activities contribute. Participation in requirements, analysis, design, and prototyping activities for third-party products is some of the most effective software work that can be done.	31
14	Storage system specifications for current platforms.	146
15	Projected storage specifications for upcoming platforms.	146
16	Status of RAJA, Umpire, and CHAI support for exascale platforms.	194

1. INTRODUCTION

The Exascale Computing Project (ECP) Software Technology (ST) focus area represents the key bridge between exascale systems and the scientists who are developing applications that will run on those platforms. The ECP offers a unique opportunity to build a coherent set of software, often referred to as the *software stack*, that will allow application developers to maximize their ability to write highly parallel applications, targeting multiple exascale architectures with runtime environments that will provide high performance and resilience. However, applications are only useful if they can provide scientific insight, and the unprecedented data produced by these applications require a complete analysis workflow that includes new technology to scalably collect, reduce, organize, curate, and analyze the data into actionable decisions. This requires scientific computing to be approached in a holistic manner, encompassing the entire user workflow from problem conception to setting up the problem with validated inputs, performing high-fidelity simulations, to the application of uncertainty quantification to the final analysis. The software stack plan defined here aims to address all of these needs by extending current technologies to exascale, where possible, by performing the research required to create new approaches that address unique problems for which current approaches will not suffice and by deploying high-quality and robust software products on the platforms developed in the exascale systems project. The ECP ST portfolio has established a set of interdependent projects that will allow a comprehensive software stack to be researched, developed, and delivered, as summarized in Table 1.

The ECP ST is developing a software stack to meet the needs of a broad set of exascale applications. The current software portfolio covers many projects that span the areas of programming models and runtimes, development tools, mathematical libraries and frameworks, data management, analysis and visualization, and software delivery. The ECP software stack was developed from the bottom up based on application requirements and the existing software stack at US Department of Energy (DOE) high-performance computing (HPC) facilities. The portfolio comprises projects selected in two different ways.

1. Thirty-two projects funded by the DOE Office of Science (SC) as part of the Advanced Scientific Computing Research (ASCR) effort. This scope of work was selected in October 2016 via a request for information (RFI) and request for proposal (RFP) process that considered prioritized requirements. The initial collection of loosely coupled projects has been reorganized twice and is now in a form that should serve the team well as it moves to the more formal execution phases of the project. Two projects were added to address scope gaps identified through regular assessment. These are the Sake project (Section 4.3.19), which provides funding to the Trilinos [9] libraries and moved the KokkosKernels effort from CLOVER 4.3.14 project to Sake. Trilinos has support from NNSA funding to port to El Capitan, but did not have support for meeting the timeline of Frontier, which is a similar system as El Capitan, and had no funding to port to Aurora. Since several ECP applications will require Trilinos on these platforms, and the broader HPC community will expect Trilinos to be available on these platforms, the Sake subproject was created. Because KokkosKernels is organizationally aligned with Trilinos, it made sense to move KokkosKernels to the Sake project. We also added the ExaWorks (Section 4.5.9) subproject to address the growing demand for a workflow Software Development Kit. ExaWorks brings together multiple workflow development teams and their products to establish a component infrastructure that will improve quality, reduce cost and accelerate delivery of new workflow capabilities needed for emerging job execution patterns on leadership computing platforms.
2. Three DOE National Nuclear Security Administration (NNSA) Advanced Simulation and Computing (ASC)-funded projects are part of the Advanced Technology Development and Mitigation (ATDM) program, which was started in FY14 and is now in its sixth year. These projects are focused on longer term research to address the shift in computing technology to extreme, heterogeneous architectures and to advance the capabilities of NNSA ASC simulation codes.

Since the initial selection process, the ECP ST has reorganized its efforts, as described in Section 1.2.

WBS 2.3.1	Programming Models and Runtimes	Cross-platform, production-ready programming infrastructure to support the development and scaling of mission-critical software at the node and full-system levels.
WBS 2.3.2	Development Tools	A suite of tools and supporting unified infrastructure aimed at improving developer productivity across the software stack. This scope includes debuggers, profilers, and the supporting compiler infrastructure with a particular emphasis on LLVM [10] as a delivery and deployment vehicle.
WBS 2.3.3	Mathematical Libraries	Mathematical libraries and frameworks that (1) interoperate with the ECP software stack, (2) are incorporated into ECP applications, and (3) provide scalable, resilient, and numerical algorithms that facilitate efficient simulations on exascale computers.
WBS 2.3.4	Data and Visualization	Production infrastructure necessary to manage, share, and facilitate the analysis and visualization of data in support of mission-critical codes. Data analytics and visualization software that support scientific discovery and understanding, despite changes in hardware architecture and the size, scale, and complexity of simulation and performance data produced by exascale platforms.
WBS 2.3.5	Software Ecosystem and Delivery	Development and coordination of software development kits (SDKs) and the Extreme-scale Scientific Software Stack (E4S) across all ECP ST projects. Development of capabilities in Spack [1] in collaboration with NNSA's primary sponsorship. Development of SuperContainers [11] and coordination of container-based workflows across DOE computing facilities.
WBS 2.3.6	NNSA ST	Development and enhancement of open-source software capabilities that are primarily developed at Lawrence Livermore, Los Alamos, and Sandia National Laboratories. Funds for engaging open science application and software teams in the use and enhancement of these products.

Table 1: ECP ST Work Breakdown Structure (WBS), technical area, and description of scope.

1.1 BACKGROUND

Historically, the software used on supercomputers has come from three sources: computer system vendors, DOE national laboratories, and academia. Traditionally, vendors have supplied system software, such as operating systems, compilers, runtime, and system-management software. The basic system software is typically augmented by software developed by the DOE HPC facilities to fill gaps or improve system management. System software often breaks or does not perform well when there is a jump in the scale of the system.

Mathematical libraries and tools for supercomputers have traditionally been developed at universities and DOE national laboratories and ported to the new computer architectures when they are deployed. Vendors also play a role in this space by optimizing the implementations of commonly used libraries and tools for

their architectures while retaining the interfaces defined by the broader community. This approach enables compile and link time replacement to improve performance on a specific platform by using the vendor versions. Mathematical libraries and tools have been remarkably robust and have supplied some of the most impactful improvements in application performance and productivity. The challenges have been the constant adapting and tuning to rapidly changing architectures.

Programming paradigms and the associated programming environments—which include compilers, debuggers, message passing, and associated runtimes—have traditionally been developed by vendors, DOE laboratories, and universities. The same can be said for file system and storage software. Vendors are ultimately responsible for providing a programming environment and file system with the supercomputer, but vendors are often disinterested in software developed by others or investing in new ideas that have few or no users yet. Also, file system software plays a key role in overall system resilience, and the difficulty of making the file system software resilient has grown nonlinearly with the scale and complexity of the supercomputers.

In addition to the lessons learned from traditional approaches, exascale computers pose unique software challenges, including the following.

- **Extreme parallelism:** Experience has shown that software breaks at each shift in scale. Exascale systems are predicted to have a billion-way concurrency almost exclusively from discrete accelerator devices, similar to today’s GPUs. An alternate approach that uses many cores with vector units is also competitive but still requires the same approximate amount of parallelism. Because clock speeds have essentially stalled, the 1,000-fold increase in potential performance going from petascale to exascale is entirely from concurrency improvements.
- **Data movement in a deep memory hierarchy:** Data movement has been identified as a key impediment to performance and power consumption. Exascale system designs are increasing the types and layers of memory, which further challenges the software to increase data locality and reuse while reducing data movement.
- **Discrete memory and execution spaces:** The node architectures of exascale systems include host CPUs and discrete device accelerators. Programming for these systems requires the coordinated transfer of data and work between the host and device. Although some of this transfer can be managed implicitly, for the most performance-sensitive phases, the programmer typically manages host-device coordination explicitly. Much of the software transformation effort will focus on this issue.

In addition to the software challenges imposed by the scale of exascale computers, the following additional requirements push the ECP away from historical approaches of obtaining the needed software for DOE supercomputers.

- **2021 acceleration:** One of the ECP’s goals is to accelerate the development of the US exascale systems and enable the first deployment by 2021. This means that the software must be ready soon. A concerted plan that accelerates the development of the highest priority and most impactful software is needed.
- **Productivity:** Traditional supercomputer software requires considerable expertise to use. One of the ECP’s goals is to make exascale computing accessible to a wider science community than was possible with previous supercomputers. This requires developing software that improves productivity and ease of use.
- **Diversity:** There is a strong push to make software run across diverse exascale systems. Accelerator devices from NVIDIA have been available for many years, and specific host-device programming and execution applications have been successfully ported to these platforms. Exascale platforms will continue to have this kind of execution model but with different programming and runtime software stacks. Writing high-performance, portable code for these platforms will be challenging.
- **Analytics and machine learning:** Future DOE supercomputers must solve emerging data science and machine learning problems in addition to the traditional modeling and simulation applications. This will require the development of scalable, parallel analytics and machine learning software for scientific applications, much of which does not exist today.

The next section describes the approach the ECP ST used to address the exascale challenges.

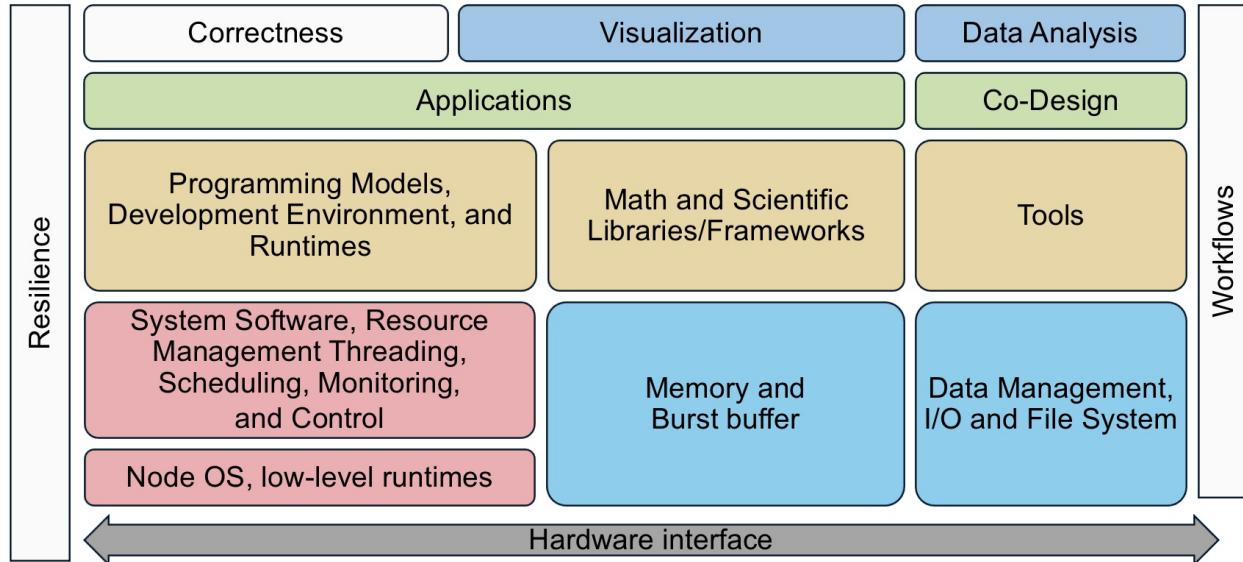


Figure 1: The ECP ST before the November 2017 reorganization. This conceptual layout emerged from several years of exascale planning conducted primarily within DOE ASCR. Because of significant restructuring of the ECP that removed many of the facility’s activities and reduced the project time line from 10 to 7 years, as well as a growing awareness of what risks had diminished, this diagram no longer represented the ECP ST efforts accurately.

1.2 ECP ST PROJECT WBS CHANGES

The initial organization of the ECP ST was based on discussions that occurred over several years of exascale planning within DOE, especially DOE ASCR. Figure 1 shows the conceptual diagram of this first phase. The 66 ECP ST projects were mapped into eight technical areas, in some cases arbitrating where a project should go based on its primary type of work, even if other work were present in the project. In November 2017, the ECP ST was reorganized into five technical areas, primarily through merging a few smaller areas, and the number of projects was reduced to 56, then to 55 because of further merging in Software Ecosystem & Delivery. Figures 2 and 3 show the diagrams of the second phase of the ECP ST. The reorganization is summarized in Figure 4. Section 2 describes the organization, planning, execution, tracking, and assessment processes that will put the ECP ST in a good position for success in the critical decision (CD) 2 phase of the project. Figure 5 shows the WBS for the ECP, and Figure 6 shows the current ST Level 2 through Level 4 WBS. The ST leadership team is shown in Figure 7.

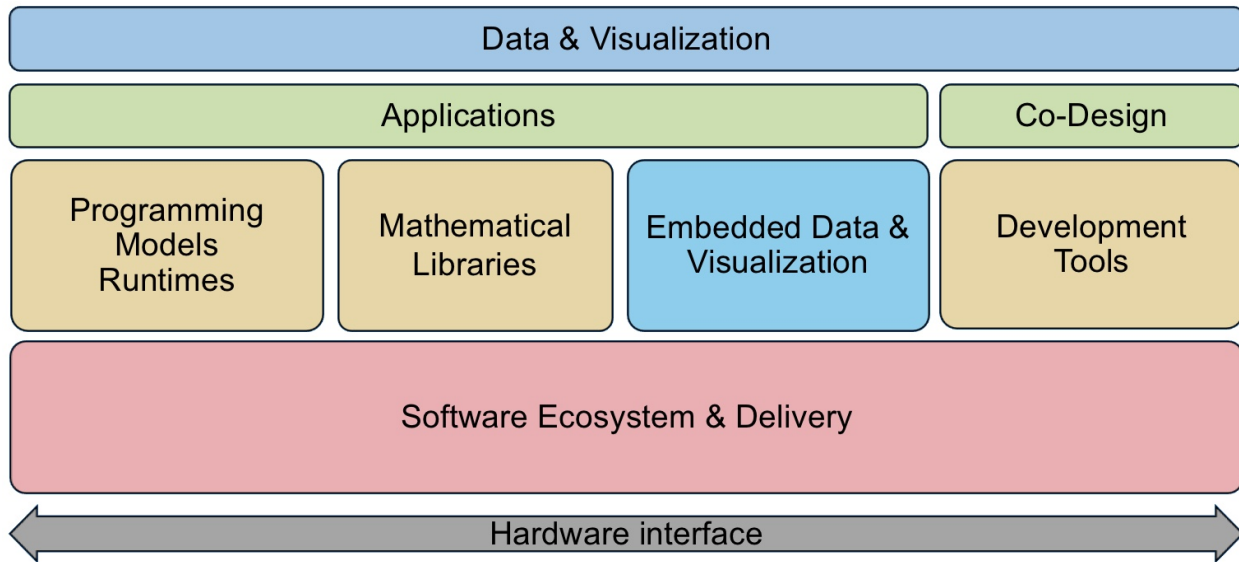


Figure 2: The ECP ST after the November 2017 reorganization. This diagram more accurately reflects the priorities and efforts of the ECP ST given the new ECP project scope and the demands foreseen.

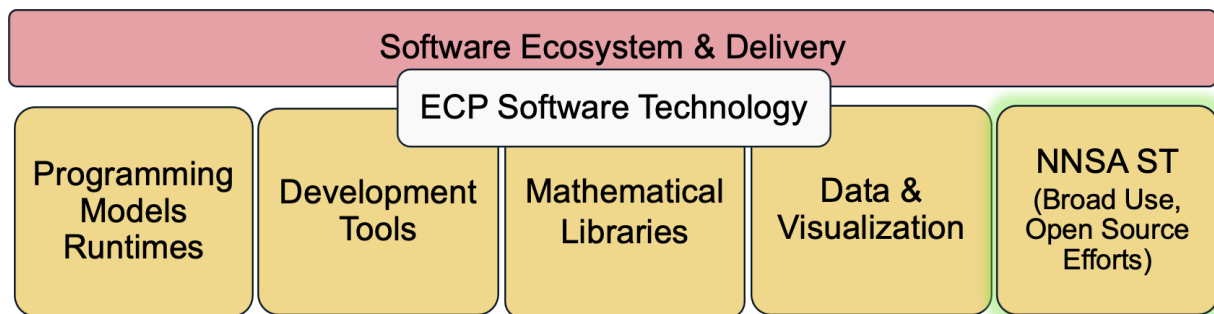


Figure 3: The ECP ST after the October 2019 reorganization. This diagram reflects the further consolidation of NNSA open-source contributions to enable the more flexible management of NNSA ST contributions.

- Phase 1: 66 total L4 subprojects
 - 35 projects funded by DOE SC that were selected in late 2016 via an RFI and RFP process, considering prioritized requirements of applications and DOE facilities. These projects started work between January–March 2017, depending on when the contracts were awarded.
 - 31 ongoing DOE NNSA-funded projects that are part of the ATDM program. The ATDM program started in FY14. These projects are focused on longer term research to address the shift in computing technology to extreme heterogeneous architectures and to advance the capabilities of NNSA simulation codes.
- Phase 2: 55 total L4 subprojects
 - 41 ASCR-funded projects. Added two Software Ecosystem & Delivery projects and four SDK projects.
 - 15 ATDM projects: Combined the previous 31 ATDM projects into one project per technical area per lab. ATDM projects are generally more vertically integrated and would not perfectly map to any proposed ECP ST technical structure. Minimizing the number of ATDM projects within the ECP WBS reduces complexity of ATDM to ECP coordination and gives ATDM flexibility in revising its portfolio without disruption to the ECP-ATDM mapping.
- Phase 3a: 33 total L4 subprojects. Fewer, larger, and more uniform-sized projects.
 - Starting in FY20, the ECP ST further consolidated L4 projects to foster additional synergies and amortize project overheads as the ECP heads into the CD-2 phase [12], in which more rigorous planning and execution are needed.
 - Five L3s to six: New NNSA ST L3.
 - 40 ST SC-funded L4 subprojects to 30.
 - * Programming Models & Runtimes: 13 to nine, Development Tools: six to six, Mathematical Libraries: seven to six, Data & Visualization: 10 to seven, Software Ecosystem & Delivery: four to three.
 - * Includes two new L4 subprojects in Software Ecosystem & Delivery.
 - 15 ST NNSA-funded projects transferred to new NNSA ST L3. Consolidated from 15 to three L4 subprojects.
 - No more small subprojects.
 - Figure 6 shows the overall structure.
- Phase 3b: 35 total L4 subprojects. Add two new L4 subprojects.
 - New L4 subproject called ExaWorks. Focuses on providing an underlying component architecture for workflow management systems and is led by a team of workflow experts who would leverage the new substrate in their own workflow products.
 - New L4 subproject called Sake. This project was created in response to a need for Trilinos funding to port to Aurora and Frontier. Concurrently, Trilinos-related activities in the CLOVER project, specifically Kokkos Kernels, were merged with the new Trilinos funding to create a more holistic project independent of CLOVER.
 - Figure 6 shows the overall structure.

Figure 4: Project remapping summary from Phase 1 (through November 2017) to Phase 2 (November 2017–September 30, 2019) to Phase 3 (After October 1, 2019).

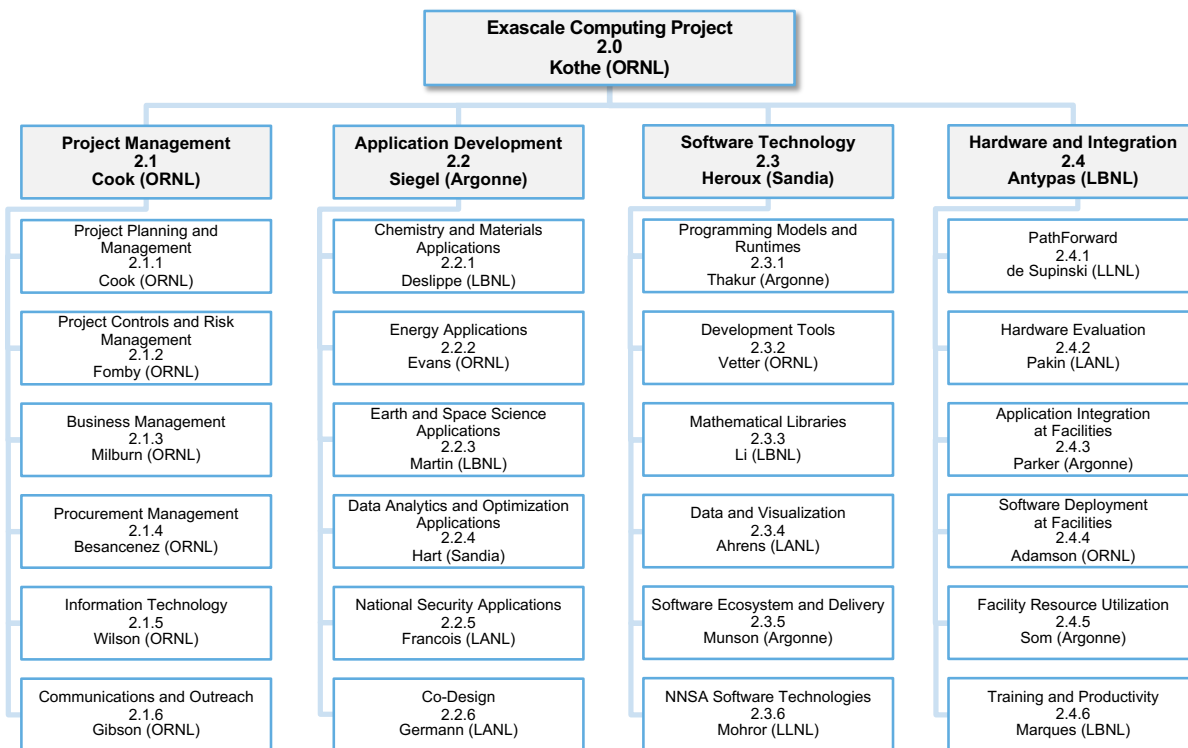


Figure 5: The ECP WBS through Level 3 (L3) as of August 2022. Under ST, WBS 2.3.6 consolidates ATDM contributions to the ECP into a new L3 area.

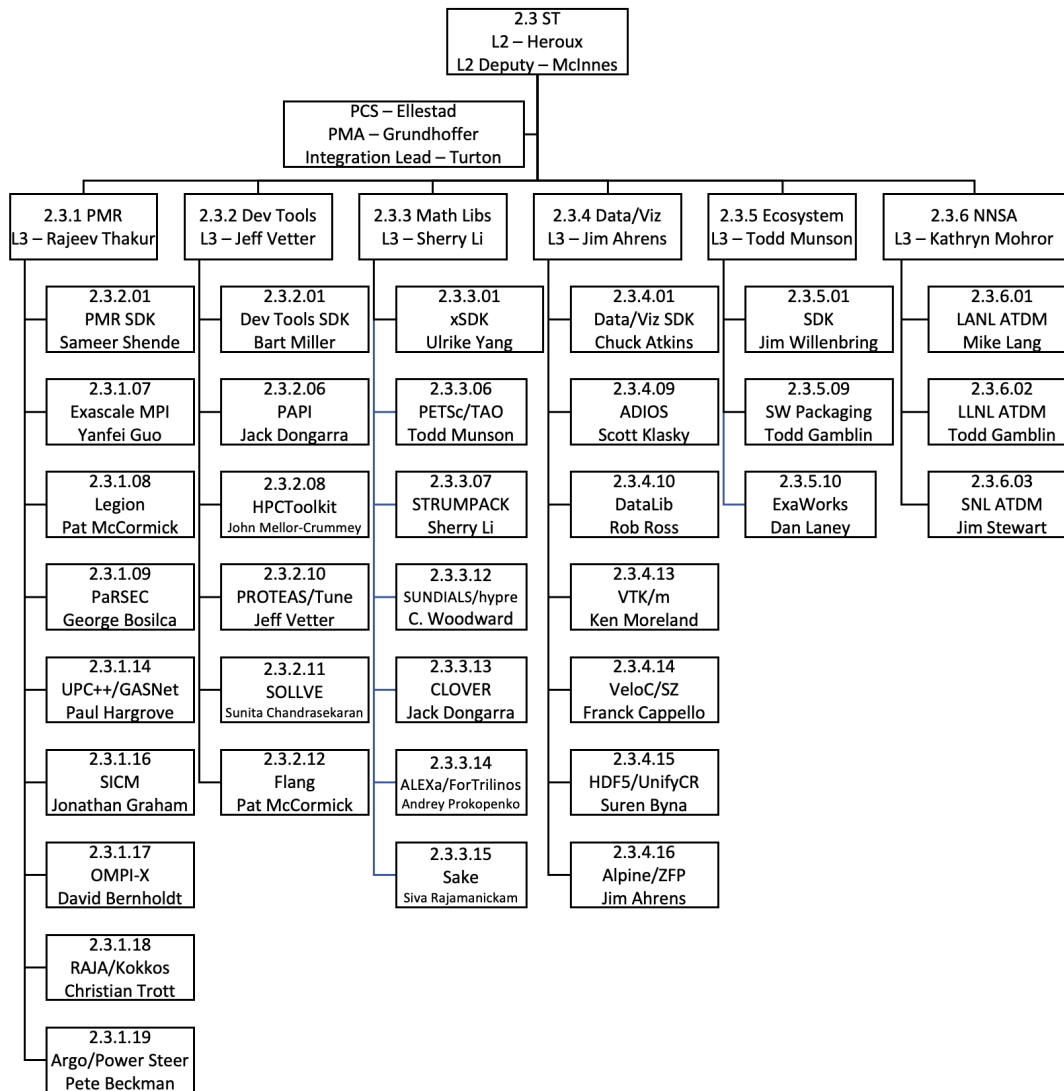
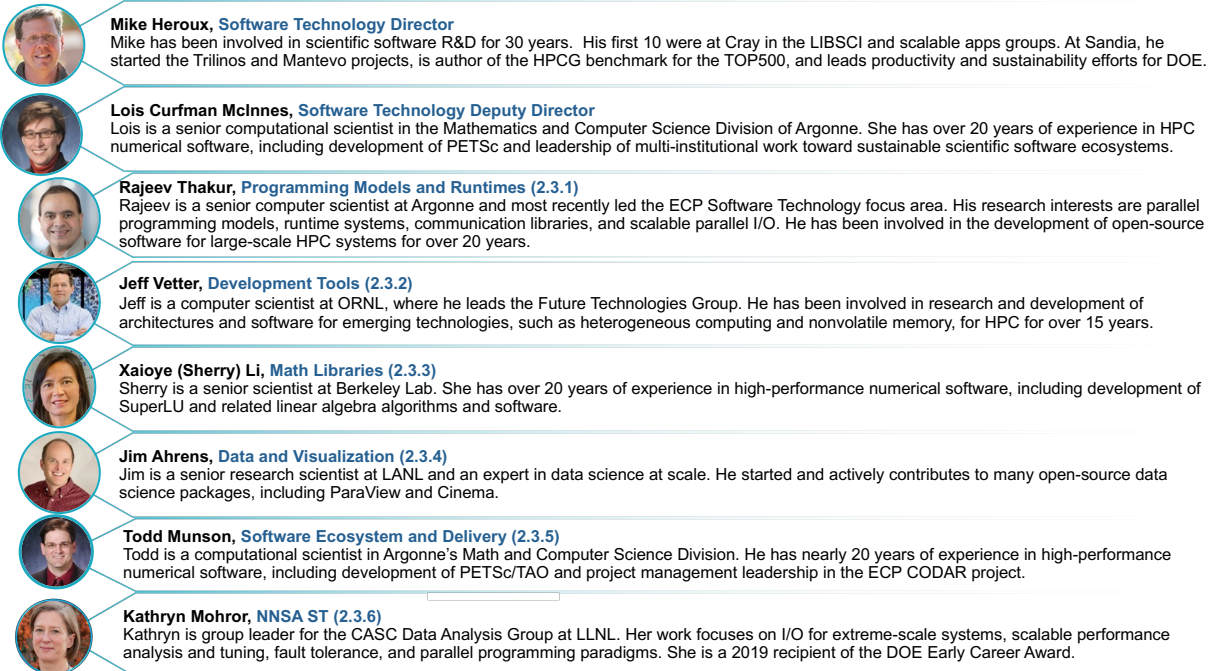


Figure 6: The FY22 ECP ST WBS as of June 1, 2022. Several teams have had PI changes since the last version of this chart. We have also added a new role of integration lead, led by Terry Turton, to work with coordination of capability integrations on our way to completing KPP-3.

ECP Software Technology Leadership Team



Mike Heroux, Software Technology Director
Mike has been involved in scientific software R&D for 30 years. His first 10 were at Cray in the LIBSCI and scalable apps groups. At Sandia, he started the Trilinos and Mantevo projects, is author of the HPCG benchmark for the TOP500, and leads productivity and sustainability efforts for DOE.

Lois Curfman McInnes, Software Technology Deputy Director
Lois is a senior computational scientist in the Mathematics and Computer Science Division of Argonne. She has over 20 years of experience in HPC numerical software, including development of PETSc and leadership of multi-institutional work toward sustainable scientific software ecosystems.

Rajeev Thakur, Programming Models and Runtimes (2.3.1)
Rajeev is a senior computer scientist at Argonne and most recently led the ECP Software Technology focus area. His research interests are parallel programming models, runtime systems, communication libraries, and scalable parallel I/O. He has been involved in the development of open-source software for large-scale HPC systems for over 20 years.

Jeff Vetter, Development Tools (2.3.2)
Jeff is a computer scientist at ORNL, where he leads the Future Technologies Group. He has been involved in research and development of architectures and software for emerging technologies, such as heterogeneous computing and nonvolatile memory, for HPC for over 15 years.

Xaioye (Sherry) Li, Math Libraries (2.3.3)
Sherry is a senior scientist at Berkeley Lab. She has over 20 years of experience in high-performance numerical software, including development of SuperLU and related linear algebra algorithms and software.

Jim Ahrens, Data and Visualization (2.3.4)
Jim is a senior research scientist at LANL and an expert in data science at scale. He started and actively contributes to many open-source data science packages, including ParaView and Cinema.

Todd Munson, Software Ecosystem and Delivery (2.3.5)
Todd is a computational scientist in Argonne's Math and Computer Science Division. He has nearly 20 years of experience in high-performance numerical software, including development of PETSc/TAO and project management leadership in the ECP CODAR project.

Kathryn Mohror, NNSA ST (2.3.6)
Kathryn is group leader for the CASC Data Analysis Group at LLNL. Her work focuses on I/O for extreme-scale systems, scalable performance analysis and tuning, fault tolerance, and parallel programming paradigms. She is a 2019 recipient of the DOE Early Career Award.

Figure 7: The ECP ST Leadership Team as of November 2020. Jonathan Carter, previous deputy director of the ECP ST, became associate laboratory director of the Computing Sciences Area at Lawrence Berkeley National Laboratory. His departure led to Lois Curfman McInnes, previously the L3 lead of Mathematical Libraries, being named the ECP ST deputy director and Sherry Li being named the new Mathematical Libraries L3 lead. Rob Neely was also promoted at Lawrence Livermore National Laboratory (LLNL), leading to Kathryn Mohror becoming the L3 lead of NNSA ST.

2. ECP ST PLANNING, EXECUTION, TRACKING AND ASSESSMENT

During the last 2 years, the ECP ST has introduced the E4S and SDKs. New approaches have been established for project planning, execution, tracking, and assessment by using a tailored earned value management system that enables iterative and incremental refinement to its planning process. The team has also revised its key performance parameter (KPP) to be solely focused on measuring capability integration into client environments. An assessment process was developed and used that has led to significant changes in the number and scope of L4 subprojects.

2.1 ECP ST ARCHITECTURE AND DESIGN

The ECP is taking an approach of co-design across all its principal technical areas: Application Development (AD), ST, and Hardware and Integration (HI). For the ECP ST, this means that its requirements are based on input from other areas, and there is a tight integration of the software products within the software stack and with applications and the evolving hardware.

The ECP ST portfolio of projects is intended to address the aforementioned exascale challenges and requirements. The ECP is not developing the entire software stack for an exascale system. For example, vendors are expected to provide the core software that comes with the system—in many cases, by leveraging ECP and other open-source efforts. Examples of vendor-provided software include operating systems; file systems; compilers for C, C++, Fortran, and so on, increasingly derived from the LLVM community ecosystem to which the ECP contributes; basic math libraries; system monitoring tools; schedulers; debuggers; vendor performance tools; MPI based on ECP-funded projects; OpenMP with features from ECP-funded projects; and data-centric stack components. The ECP develops other, mostly higher level software that is needed by applications and is not vendor specific. ECP-funded software activities are concerned with extreme scalability, exposing additional parallelism, unique requirements of exascale hardware, and performance-critical components. Other software that aids in developer productivity is needed and could come from third-party, open-source efforts.

The ST portfolio includes both ASCR and NNSA ATDM-funded efforts. The memorandum of understanding established between DOE SC and NNSA has formalized this effort. Whenever possible, ASCR and ATDM efforts are treated uniformly in ECP ST planning and assessment activities.

ST also plans to increase integration within the ST portfolio through the increased use of software components and application composition vs. monolithic application design. One important transition that ECP can accelerate is the increased development and delivery of reusable scientific software components and libraries. Although math and scientific libraries have long been a successful element of the scientific software community, their use can be expanded to include other algorithms and software capabilities so that applications can be considered more of an aggregate composition of reusable components than a monolithic code that uses libraries tangentially.

To accelerate this transition, a greater commitment on the part of software component developers is needed to provide reliable and portable software that users can consider to be part of the software ecosystem in much the same way users depend on MPI and compilers. However, application developers must be expected to participate as clients and users of reusable components, using capabilities from components, transitioning away from—or keeping a backup option of—their own custom capabilities.

2.1.1 *The Extreme-Scale Scientific Software Stack*

In October 2020, the ECP ST released V1.2 of the E4S.¹ E4S contains a collection of the software products to which ECP ST contributes. E4S is the primary conduit for providing easy access to ECP ST capabilities for the ECP and the broader community. E4S is also the ECP ST vehicle for regression and integration testing across DOE pre-exascale and exascale systems.

¹<http://e4s.io>

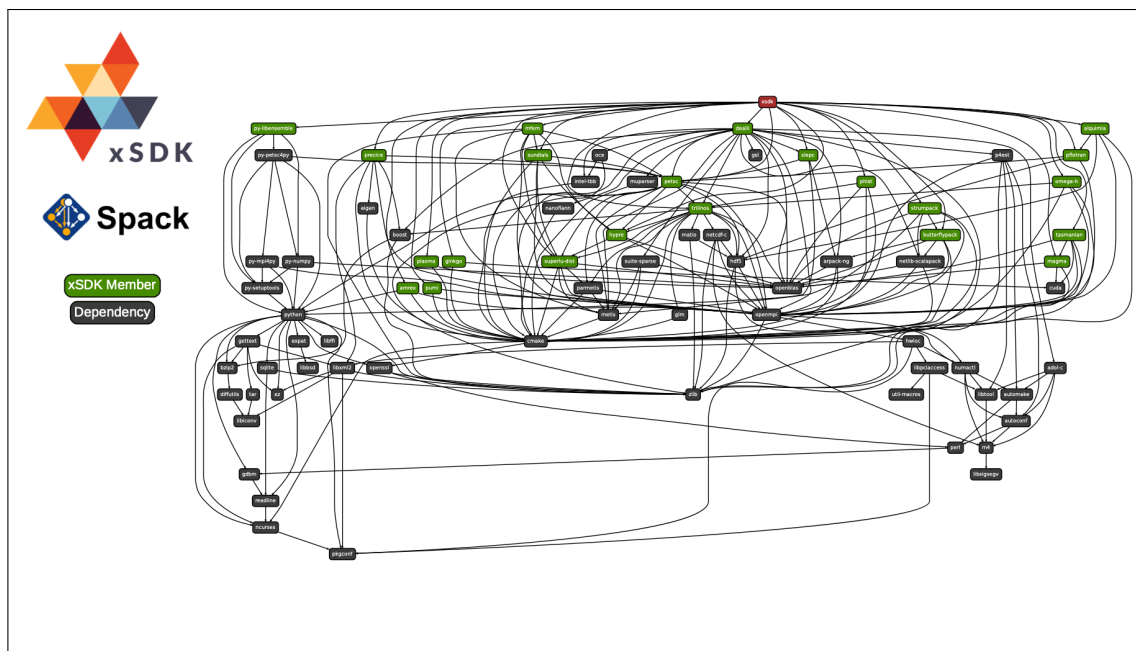


Figure 8: Using Spack [2], E4S builds a comprehensive software stack. As ECP ST efforts proceed, E4S will be used for continuous integration testing, providing developers with rapid feedback on regression errors and providing user facilities with a stable software base as the team prepares for exascale platforms. This diagram shows how E4S builds ECP products via an SDK target, which are the math libraries’ SDK, called xSDK in this example. The SDK target then builds all products that are part of the SDK (see Figure 12 for SDK groupings), first defining and building external software products. Green-labeled products are part of the SDK. The blue label indicates expected system tools, which, in this case, is a particular version of Python. Black-labeled products are expected to be previously installed into the environment, which is a common and easily satisfied requirement. Using this approach, users interested in only SUNDIALS, a particular math library, can be assured that the SUNDIALS build will be possible because it is a portion of what E4S builds and tests.

E4S has the following key features.

- **The E4S suite is a large and growing effort to build and test a comprehensive scientific software ecosystem.** In November 2018, E4S V0.1 contained 25 ECP products. Two years later, E4S V1.2, the fifth E4S release, contained 67 ECP ST products and numerous additional products needed for a complete software environment. Eventually, E4S will contain all the open-source products to which ECP contributes and all the related products needed for a holistic environment.
- **E4S is not an ECP-specific software suite.** The products in E4S represent a holistic collection of capabilities that contain the ever-growing SDK collections sponsored by the ECP and all additional underlying software required to use ECP ST capabilities. Furthermore, the E4S effort is expected to live beyond the time span of the ECP, becoming a critical element of the scientific software ecosystem.
- **E4S is partitionable.** E4S products are built and tested together by using a tree-based hierarchical build process. Because the entire E4S tree is built and tested, users can build any subtree of interest without building the whole stack (Figure 8).
- **E4S uses Spack.** The Spack [2] meta-build tool invokes the native build process of each product, enabling the quick integration of new products, including non-ECP products. E4S packages are available as pre-built Spack binaries in the E4S Build Cache (Figure 9). Binaries for the major operating systems and architectures are added to the build cache as updates become available in Spack.

- **E4S is available via containers and cloud environments.** In addition to a build-from-source capability using Spack, E4S maintains several container environments (e.g., Docker, Singularity, Shifter, CharlieCloud) that provide the lowest barrier to use. Container distributions dramatically reduce installation costs and provide a ready-made environment for tutorials that leverage E4S capabilities. For example, E4S containers now support custom images for ECP applications, such as WDMapp and Pantheon. E4S is also available via AWS and Google Cloud platforms. (Figure 10)
- **E4S distribution.** E4S products are available at the E4S website.²
- **E4S developer community resources.** Developers interested in participating in E4S can visit the E4S-Project GitHub community.³

The first set of E4S community policies [13] was adopted in October 2020 (Figure 11). These policies are membership criteria for a product to become an E4S member package. The purpose of the community policies is to establish baseline software quality and practice expectations to help address sustainability and interoperability challenges for the ST software ecosystem. Although a package does not have to demonstrate compatibility with the policies as a condition of inclusion in E4S releases, compatibility is necessary for member package designation.

The E4S effort is described in further detail in Sections 2.1.2 and 4.5. Table 2 summarizes what E4S is, and is not.

²<http://e4s.io>

³<https://github.com/E4S-Project>

E4S Build Cache for Spack 0.18.0

To add this mirror to your Spack:

```
$> spack mirror add E4S https://cache.e4s.io
$> spack buildcache keys -it
```

88,401 total packages

Last updated 2022-05-30 16:42 PDT

All OS
 PPC64LE
 X86_64
 CentOS 7
 CentOS 8
 RHEL 7
 RHEL 8
 Ubuntu 18.04
 Ubuntu 20.04

Search

[adiak@0.1.1](#)
[adiak@0.2.1](#)
[adios2@2.5.0](#)
[adios2@2.6.0](#)
[adios2@2.7.0](#)
[adios2@2.7.1](#)
[adios2@2.8.0](#)
[adios@1.13.1](#)
[adlbc@0.9.2](#)
[adlbc@1.0.0](#)
[adol-c@2.7.2](#)
[alquimia@1.0.9](#)
[alsa-lib@1.2.3.2](#)
[amg@1.2](#)
[aml@0.1.0](#)
[amr-wind@ascent](#)
[amr-wind@main](#)
[amrex@20.07](#)
[amrex@20.09](#)
[amrex@20.10](#)
[amrex@20.11](#)
[amrex@20.12](#)
[amrex@21.01](#)
[amrex@21.02](#)
[amrex@21.03](#)
[amrex@21.04](#)
[amrex@21.05](#)
[amrex@21.06](#)
[amrex@21.07](#)
[amrex@21.08](#)
[amrex@21.09](#)
[amrex@21.10](#)
[amrex@21.11](#)
[amrex@21.12](#)
[amrex@22.01](#)
[amrex@22.02](#)
[amrex@22.03](#)
[amrex@22.04](#)
[amrex@22.05](#)
[ant@1.10.0](#)
[ant@1.10.7](#)
[antlr@2.7.7](#)
[arborx@0.9-beta](#)
[arborx@1.0](#)
[arborx@1.1](#)
[arborx@1.2](#)
[archer@2.0.0](#)
[argobots@1.0](#)
[argobots@1.0rc1](#)
[argobots@1.0rc2](#)
[argobots@1.1](#)
[arpack-ng@3.7.0](#)
[arpack-ng@3.8.0](#)
[ascent@0.6.0](#)
[ascent@0.7.0](#)
[ascent@0.7.1](#)
[ascent@0.8.0](#)
[ascent@develop](#)
[ascent@pantheon_ver](#)
[asio@1.16.1](#)
[asio@1.18.2](#)
[asio@1.20.0](#)
[asio@1.21.0](#)
[assimp@4.0.1](#)
[assimp@5.0.1](#)
[assimp@5.1.4](#)
[assimp@5.2.2](#)
[assimp@5.2.3](#)
[at-spi2-atk@2.38.0](#)
[at-spi2-core@2.40.1](#)
[atk@2.36.0](#)
[autoconf-archive@2019.01.06](#)
[autoconf-archive@2022.02.11](#)
[autoconf@2.69](#)
[autoconf@2.70](#)
[automake@1.15.1](#)
[automake@1.16.1](#)
[automake@1.16.2](#)
[automake@1.16.3](#)
[automake@1.16.5](#)
[axl@0.1.1](#)
[axl@0.3.0](#)
[axl@0.4.0](#)
[axl@0.5.0](#)
[axom@0.3.3](#)
[axom@0.4.0](#)
[axom@0.5.0](#)
[axom@0.6.1](#)
[bacio@2.4.1](#)
[bash@5.0](#)
[bats@0.4.0](#)
[bdfopcf@1.0.5](#)
[berkeley-db@18.1.40](#)
[berkeley-db@6.2.32](#)
[binutils@2.31.1](#)
[binutils@2.32](#)
[binutils@2.33.1](#)
[binutils@2.34](#)
[binutils@2.36.1](#)
[binutils@2.37](#)
[binutils@2.38](#)
[bison@3.4.2](#)
[bison@3.6.4](#)
[bison@3.7.4](#)
[bison@3.7.6](#)
[bison@3.8.2](#)
[blaspp@2020.10.02](#)
[blaspp@2021.04.01](#)
[blt@0.3.6](#)
[blt@0.3.6rcm](#)
[blt@0.4.0](#)
[blt@0.4.1](#)
[blt@0.5.0](#)
[blt@develop](#)
[bmi@develop](#)
[bmi@main](#)
[bolt@1.0.2](#)
[bolt@1.0rc2](#)
[bolt@1.0rc3](#)
[bolt@2.0](#)
[boost@1.68.0](#)
[boost@1.70.0](#)
[boost@1.72.0](#)
[boost@1.73.0](#)
[boost@1.74.0](#)
[boost@1.75.0](#)
[boost@1.76.0](#)
[boost@1.77.0](#)
[boost@1.78.0](#)
[boost@1.79.0](#)
[bricks@r0.1](#)
[bufpr@11.5.0](#)
[butterflypack@1.1.0](#)
[butterflypack@1.2.0](#)
[butterflypack@1.2.1](#)
[butterflypack@2.0.0](#)
[butterflypack@2.1.0](#)
[butterflypack@2.1.1](#)
[bvacc@master](#)
[bzip2@1.0.6](#)
[bzip2@1.0.8](#)
[c-ares@1.15.0](#)
[c-blosc@1.17.0](#)
[c-blosc@1.21.0](#)
[c-blosc@1.21.1](#)
[cabana@0.3.0](#)
[cabana@0.4.0](#)
[cairo@1.16.0](#)
[caliper@2.0.1](#)
[caliper@2.2.0](#)
[caliper@2.3.0](#)
[caliper@2.4.0](#)
[caliper@2.5.0](#)
[caliper@2.6.0](#)
[caliper@2.7.0](#)
[camp@0.1.0](#)
[camp@0.2.2](#)
[camtimers@master](#)
[catalyst@5.6.0](#)
[cdo@1.9.10](#)
[cereal@1.3.2](#)
[cgns@4.2.0](#)
[chai@2.3.0](#)
[chai@2.4.0](#)
[charliecloud@0.22](#)
[charliecloud@0.23](#)
[charliecloud@0.24](#)
[charliecloud@0.25](#)
[charliecloud@0.26](#)
[cinch@develop](#)
[cinch@master](#)
[cli11@1.9.1](#)
[cmake@3.13.4](#)
[cmake@3.14.5](#)
[cmake@3.14.7](#)
[cmake@3.15.4](#)
[cmake@3.16.2](#)
[cmake@3.16.5](#)
[cmake@3.17.1](#)
[cmake@3.17.3](#)
[cmake@3.18.0](#)
[cmake@3.18.1](#)
[cmake@3.18.2](#)
[cmake@3.18.4](#)
[cmake@3.19.0](#)
[cmake@3.19.2](#)
[cmake@3.19.5](#)
[cmake@3.19.7](#)
[cmake@3.20.0](#)
[cmake@3.20.1](#)
[cmake@3.20.2](#)
[cmake@3.20.3](#)
[cmake@3.20.5](#)
[cmake@3.20.6](#)
[cmake@3.21.2](#)
[cmake@3.21.3](#)
[cmake@3.21.4](#)
[cmake@3.22.1](#)
[cmake@3.22.2](#)
[cmake@3.22.3](#)
[cmake@3.23.0](#)
[cmake@3.23.1](#)
[codar-cheetah@develop](#)
[comgr@3.9.0](#)
[comgr@4.0.0](#)
[comgr@4.1.0](#)
[comgr@4.2.0](#)
[comgr@4.3.0](#)
[comgr@4.3.1](#)
[comgr@4.5.0](#)
[comgr@4.5.2](#)
[comgr@5.0.2](#)
[comgr@5.1.0](#)
[compositeproto@0.4.2](#)
[conduit@0.6.0](#)
[conduit@0.7.1](#)
[conduit@0.7.2](#)
[conduit@0.8.0](#)
[conduit@0.8.1](#)
[conduit@0.8.2](#)
[conduit@0.8.3](#)
[conduit@develop](#)
[conduit@master](#)
[coupler@master](#)
[cpio@2.13](#)
[crtm@2.3.0](#)
[cub@1.12.0-rc0](#)
[cub@1.16.0](#)
[cube@4.6](#)
[cubelib@4.6](#)
[cubew@4.5](#)
[cuda@10.1.243](#)
[cuda@10.2.89](#)
[cuda@11.0.2](#)
[cuda@11.0.3](#)
[cuda@11.1.0](#)
[cuda@11.1.1](#)
[cuda@11.1.1](#)
[cuda@11.2.0](#)
[cuda@11.2.1](#)
[cuda@11.2.2](#)
[cuda@11.3.0](#)
[cuda@11.3.1](#)
[cuda@11.4.0](#)
[cuda@11.4.2](#)
[cuda@11.5.0](#)
[cuda@11.5.1](#)
[cuda@11.6.0](#)
[cuda@11.6.1](#)
[cuda@11.7.0](#)
[curl@7.63.0](#)
[curl@7.71.0](#)
[curl@7.72.0](#)
[curl@7.73.0](#)
[curl@7.74.0](#)
[curl@7.75.0](#)
[curl@7.76.0](#)
[curl@7.76.1](#)
[curl@7.78.0](#)
[curl@7.79.0](#)
[curl@7.80.0](#)
[curl@7.81.0](#)
[curl@7.82.0](#)
[curl@7.83.0](#)
[cymq@4.1.1](#)
[damageproto@1.2.1](#)
[darshan-runtime@3.1.7](#)
[darshan-runtime@3.1.8](#)
[darshan-runtime@3.2.1](#)
[darshan-runtime@3.3.0](#)
[darshan-runtime@3.3.1](#)

Figure 9: Using Spack build cache features, E4S builds can be accelerated via cached binaries for any build signature that Spack has already seen. Between September 2020 and June 2022, more than 42,000 binaries were added to the cache.

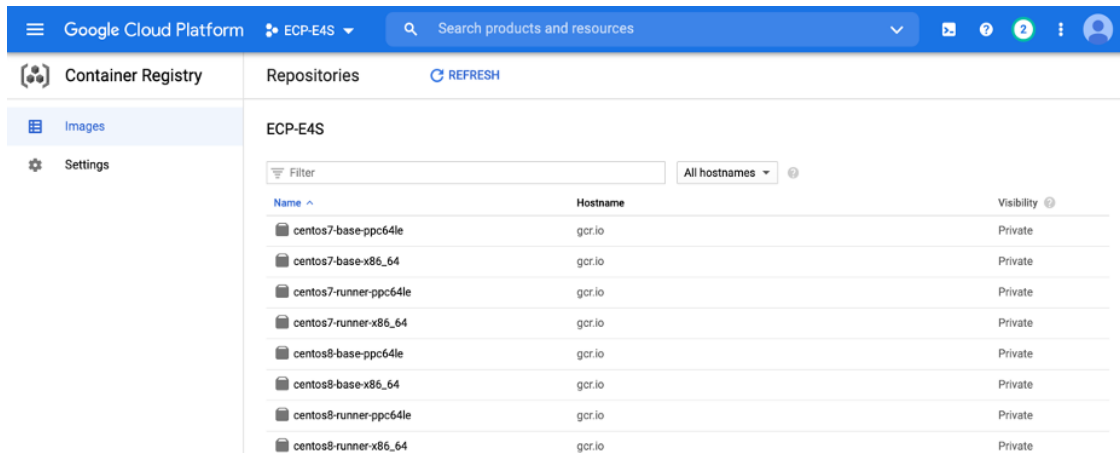


Figure 10: E4S now supports Google Cloud Platform in addition to Amazon Web Services (AWS).

What E4S is not	What E4S is
A closed system taking contributions only from DOE software development teams	An extensible, open architecture software ecosystem that accepts contributions from US and international teams A framework for collaborative open-source product integration
A monolithic, take-it-or-leave-it software behemoth	A full collection of compatible software capabilities A manifest of a la carte, selectable software capabilities
A commercial product	A vehicle for delivering high-quality, reusable software products in collaboration with others
A simple repackaging of existing software	The conduit for future-leading edge HPC software that targets scalable, next-generation computing platforms A hierarchical software framework to enhance (via SDKs) software interoperability and quality expectations

Table 2: E4S provides a complete Linux-based software stack that is suitable for many scientific workloads, tutorial, and development environments. Additionally, it is an open-software architecture that can expand to include any additional and compatible Spack-enabled software capabilities. Because Spack packages are available for many products and easily created for others, E4S is practically expandable to include almost any robust Linux-based product. Furthermore, E4S capabilities are available as subtrees of the full build. E4S is not monolithic.

P1: Spack-Based Build and Installation Each E4S member package supports a scriptable Spack build and production-quality installation in a way that is compatible with other E4S member packages within the same environment. When E4S build, test, or installation issues arise, it is expected that the team will collaboratively resolve those issues.

P2: Minimal Validation Testing Each E4S member package has at least one test that is executable through the E4S validation suite (<https://github.com/E4S-Project/testsuite>). This will be a post-installation test that validates the usability of the package. The E4S validation test suite provides basic confidence that a user can compile, install, and run every E4S member package. The E4S team can actively participate in the addition of new packages to the suite upon request.

P3: Sustainability All E4S compatibility changes will be sustainable in that no changes go into the regular development and release versions of the package and should not be in a private release/branch that is provided only for E4S releases.

P4: Documentation Each E4S member package should have sufficient documentation to support installation and use.

P5: Product Metadata Each E4S member package team will provide key product information via metadata that is organized in the E4S DocPortal format. Depending on the file names for the metadata, this may require minimal setup.

P6: Public Repository Each E4S member package will have a public repository (e.g., GitHub, Bitbucket) where the development version of the package is available and pull requests can be submitted.

P7: Imported Software If an E4S member package imports software that is externally developed and maintained, then it must allow installing, building, and linking against a functionally equivalent outside copy of that software. Acceptable ways to accomplish this include (1) forsaking the internal copied version and using an externally provided implementation or (2) changing the file names and name spaces of all global symbols to allow the internal copy and the external copy to coexist in the same downstream libraries and programs. This pertains primarily to third-party support libraries and does not apply to key components of the package that may be independent packages but are also integral components of the package itself.

P8: Error Handling Each E4S member package will adopt and document a consistent system for signifying error conditions as appropriate for the language and application (e.g., returning an error condition or throwing an exception). In the case of a command line tool, it should return a sensible exit status on success and failure so the package can safely run from within a script.

P9: Test Suite Each E4S member package will provide a test suite that does not require special system privileges or the purchase of commercial software. This test suite should grow in its comprehensiveness over time (i.e., new and modified features should be included in the suite).

Figure 11: Version 1 of the E4S community policies. These policies serve as membership criteria for E4S member packages. The E4S community policy effort heavily leveraged the existing xSDK community policies [3].

2.1.2 Software Development Kits

One opportunity for a large software ecosystem project such as the ECP ST is to foster increased collaboration, integration, and interoperability among its funded efforts. Part of the ECP ST design is the creation of SDKs.

SDKs are collections of related software products, called *packages*, in which coordination across package teams will improve usability and practices and foster community growth among teams that develop similar and complementary capabilities. SDKs have the following attributes.

1. **Domain scope:** Each SDK will comprise packages whose capabilities are within a natural functionality domain. Packages within an SDK provide similar capabilities that can enable leveraging of common requirements, design, testing, and similar activities. Packages could have a tight complementarity so that ready composability is valuable to the user.
2. **Interaction models:** This is how packages within an SDK interact with each other. Packages may be compatible, complementary, and/or interoperable. Interoperability includes common data infrastructure, or the seamless integration of other data infrastructures, and access to capabilities from one package for use in another.
3. **Community policies:** These include expectations for how package teams will conduct activities, the services they provide, the software standards they follow, and other practices that can be commonly expected from a package in the SDK.
4. **Meta-build system:** This includes robust tools and processes to build from source, install, and test the SDK with compatible versions of each package. This system sits on top of the existing build, install, and test capabilities for each package.
5. **Coordinated plans:** Development plans for each package will include efforts to improve SDK capabilities and lead to better integration and interoperability.
6. **Community outreach:** Efforts to reach out to the user and client communities will include explicit focus on SDK as a product suite.

SDKs provide an aggregation of software products that have complementary or similar attributes. The ECP ST uses SDKs to better ensure product interoperability and compatibility. SDKs are also essential aggregation points for coordinated planning and testing. SDKs are an integral element of the ECP ST [14]. Section 4.5.7 describes the six SDK groupings and the current status of the SDK effort.

ECP ST SDKs As part of the delivery of ECP ST capabilities, the team will establish and grow a collection of SDKs. The new layer of aggregation that SDKs represent is important for improving all aspects of product development and delivery. The communities that will emerge from SDK efforts will lead to better collaboration and higher quality products. Established community policies will provide a means to grow SDKs beyond the ECP to include any relevant external effort. The meta-build configurations based on Spack will play an important role in managing the complexity of building the ECP ST software stack by providing a new layer in which versioning, consistency, testing, and build options management can be addressed at a mid-scope below the global build of ECP ST products. Each of the five ECP ST L3 areas has an SDK project. These projects shepherd five of the six SDKs, along with the products not associated with an SDK. The other SDK, the new Computational Workflows SDK, is overseen by the ExaWorks project. Section 4.5.7 provides an update on the progress in defining SDK groupings. For visibility, the same diagram is provided in Figure 12.

2.1.3 ECP ST Product Dictionary

In the last year, the ECP has initiated an effort to explicitly manage ECP ST products and their dependencies (Section 2.1.4). To eliminate ambiguities, a product dictionary—an official list of publicly named products to which ECP ST teams contribute their capabilities and upon which ECP ST clients depend—is needed. The ECP product dictionary is one managed table that presently contains 70 primary products and subproducts that are either components within a product or particular implementations of a standard application programming interface (API). Two special primary products are the Facilities stack and Vendor stack. Having these stacks on the list enables ST teams to indicate that the team’s capabilities are being delivered to ecosystems outside of the ECP.

ECP ST Product SDK Assignments - Version 2						
xSDK* (19)	PMR Core (23)	Tools & Technology (13)	Compilers & Support (8)	Data & Visualization (16)	Computational Workflows (6)	Ecosystem/E4S at-large (3)
hypr	Legion	TAU	<i>openarc</i>	ParaView	<i>Flux</i>	Spack
MFEM	Kokkos	HPCToolkit	<i>Kitsune</i>	<i>Catalyst</i>	<i>Parsl</i>	mpiFileUtils
Kokkoskernels	RAJA	Dyninst Binary Tools	LLVM	VTK-m	<i>Swift/T</i>	Charlecloud
Trilinos	CHAI	Caliper	<i>CHILL Autotuning Compiler</i>	SZ	<i>RADICAL Cybertools</i>	
SUNDIALS	PaRSEC	PAPI	<i>LLVM OpenMP compiler</i>	zfp	<i>BEE</i>	
PETSc/TAO	<i>DARMA</i>	<i>Program Database Toolkit</i>	<i>OpenMP V & V</i>	<i>VisIt</i>	<i>Balsam</i>	
py-libEnsemble	GASNet-EX	<i>TiMemory</i>	Flang/LLVM Fortran compiler	ASCENT		
STRUMPACK	Qthreads	ReMPI	<i>Clacc</i>	Cinema		
SuperLU	BOLT	NINJA		<i>SENSEI</i>		
ForTrilinos	UPC++	<i>STAT</i>		HDF5		
SLATE	MPICH	Archer		Parallel netCDF		
MAGMA	Open MPI	FLIT		ADIOS		
DTK	Umpire	<i>FPUChecker</i>		Darshan		
Tasmanian	<i>QUO</i>			UnifyFS		
Ginkgo	Papyrus			VeloC		
PLASMA	<i>SICM</i>			SCR		
HeFFTe	AML					
AMReX	Mercury					
ArborX	FAODEL					
	FileCSI					
	<i>PowerStack</i>					
	UMap					
	NRM					

Key
 Not included in E4S 2021.08
 * All products in xSDK except ForTrilinos and ArborX have formally adopted SDK-specific community policies

Figure 12: The breakdown of ECP ST products into six SDKs are shown in the first six columns. The rightmost column lists products that are not part of an SDK but are part of the Ecosystem group that will also be delivered as part of the E4S. Section 4.5.7 provides an update on the progress in defining SDK groupings.

Figure 13 describes the policy for ECP ST teams to add and manage their contributions to the product dictionary. Figure 14 shows a snapshot of the beginning and end of the current *ECP ST Product Dictionary*, which is maintained on the ECP Confluence wiki.

2.1.4 ECP Product Dependency Management

Given the *ECP ST Product Dictionary* and a similar dictionary for ECP AD and co-design products, the ECP has created a dependency database that enables the creation and characterization of product-to-product dependencies. The ECP manages these dependencies in a Jira database by using a custom Jira issue type: Dependency. The dependency database provides an important tool for understanding and managing ECP activities. The dependency information is valuable within and outside the project. Figure 15 shows a screenshot of the Jira dependency database dashboard, and Figure 16 shows a screenshot of an example query result in the Jira dependency database.

ECP ST Product Dictionary

Created by Mike Heroux, last modified by Vivek Kale on 2019-10-05

The ECP Software Technology (ST) Product Dictionary is the official list of publicly recognized names to which ECP ST efforts contribute. While ST teams use an expanded product namespace, the list on this page indicates the eventual access point for ST product development efforts.

This table lists in **bold** only those products that are typically recognizable to users. We call these **primary products**. Examples:

1. MPI is commonly known by users. MPICH and OpenMPI both provide implementations of that product.
2. Fortran is a product. Flang is a particular Fortran product. LLVM is a backend for some Fortran compilers.
3. FFT is a product. FFTX, FFT-ECP provide FFT capabilities through interchangeable interfaces.
4. C++ is a product. Clacc provides capabilities for Clang, as does LLVM.

Subproducts are listed underneath with the primary product name as a prefix.

In some cases, a product may be listed as a primary product even if it is not widely recognized by the user community. In this case, the product developer needs to be able to address these additional criteria:

1. The product is intended for stand-alone delivery and not be included in some other product in the future.
2. There is a credible sustainability path for the product and the development team provides some evidence that it can support the product in a sustainable way, including staffing and funding.

Deployment scope is meant to give others a sense of how widely integrated a product is in the HPC ecosystem and how far along it is in maturing toward usability. The deployment scope categories are:

- **Broad:** Widely used in the HPC ecosystem, expected to be available and usable by many applications.
- **Moderate:** Some use in applications but not ubiquitous.
- **Experimental:** Still under development and used by collaborators and friendly users.

Figure 13: Screenshot from the top of the ECP Confluence wiki page containing the *ECP ST Product Dictionary*. The product dictionary structure contains primary and secondary products. Client (i.e., consumer) dependencies are stated against the primary product names only, enabling unambiguous mapping of AD-on-ST and ST-on-ST dependencies.

Product	URL	Description/Notes	Deployment Scope	Technical Area	Point of Contact
1. ADIOS	https://github.com/ornladios/ADIOS2	I/O and data management library for storage I/O, in-	Broad	Data & Viz	@Scott Klasky
2. AID					
AID: STAT	https://github.com/LLN				
AID: Archer	https://github.com/PRR				
AID: FLIT	https://github.com/PRR				
66. Vendor Stack			Experimental	Software Ecosystem	TBD
67. VeloC	https://github.com/ECP-VeloC/VELOC	Scalable checkpoint-restart library.	Broad	Data & Viz	@Franck Cappello
68. VTK-m	http://m.vtk.org	Parallel on-node visualization toolkit.	Broad	Data & Viz	@Kenneth Moreland
69. xSDK	https://xsdk.info	Math libraries meta product combining the most popular HPC math libraries into a compatible collection.	Moderate	Math libraries	@Ulrike Meier Yang
70. zfp	https://github.com/LLNL/zfp	In-memory data compression library.	Moderate	Data & Viz	@Peter Lindstrom @Terry Turton

Figure 14: Screenshots of the *ECP ST Product Dictionary* table. The table is actively managed to include primary and secondary products to which the ECP ST team contributes and upon which ECP ST clients depend. Presently, the product dictionary contains 70 primary products. Secondary products are listed under the primary product with the primary product as a prefix. For example, AID is the second primary product in the table shown here. STAT, Archer, and FLIT are component subproducts. MPI (not shown) is another primary product. MPICH and OpenMPI are two robust MPI implementations and are listed as MPI subproducts.



Figure 15: Using Jira, the ECP manages its AD, ST, HI, vendor, and facilities dependencies. This figure shows a screenshot of the Jira dependency database dashboard and an edit panel, which supports the creation and management of a consumer-on-producer dependency.

Search Save as Share Export Tools

Project: All ▼ Dependency ▼ Status: All ▼ Assignee: All ▼ Contains text More Search Advanced

1-50 of 814 Columns

T	Key	Summary	Assignee	Reporter	Status ↑	Resolution	Created	Due	Baseline end date	Links	Development
🔗	INT-691	PETSc/TAO ↔ Spack	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				
🔗	INT-686	HDF5 ↔ PETSc/TAO	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				
🔗	INT-690	PETSc/TAO ↔ xSDK	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				
🔗	INT-687	MPI-IO ↔ PETSc/TAO	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				
🔗	INT-689	OpenMP ↔ PETSc/TAO	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				
🔗	INT-677	libEnsemble ↔ xSDK	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				
🔗	INT-681	BLAS ↔ PETSc/TAO	Unassigned	Todd Munson	APPROVED	Approved	2019-09-10				

Figure 16: This query result from the ECP Jira dependency database lists all consumers of capabilities from the PETSc/TAO product. Selecting the details of one of the dependency issues shows how critical the dependency is and any custom information peculiar to the particular dependency.

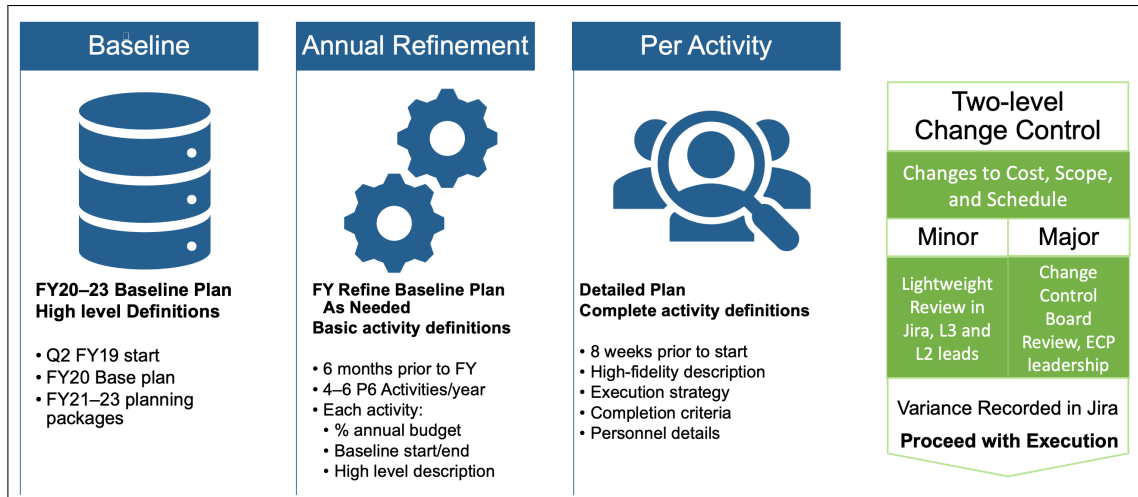


Figure 17: Example of a P6 Activity.

2.2 ECP ST PLANNING AND TRACKING

Although the ECP is an official 413.3B federal project using an earned value management system, the ECP is permitted to tailor the planning process to obtain the flexibility needed for a software project, the requirements of which are emerging as the project proceeds. This section describes how the ECP ST plans activities using the Jira project management tool. The following sections discuss P6 activities, which are similar to milestones, and KPP-3, which is associated with the ECP ST.

2.2.1 ECP ST P6 Activity Issues

ECP ST uses a custom Jira issue type called *P6 Activity*. Each L4 subproject creates a series of P6 activity issues that extend to the end of the ECP (Q3 FY23). Except for the current fiscal year, one P6 Activity issue describes expected deliverables as a planning package. Six months before the start of a fiscal year, the planning package for the coming year is replaced with four to six issues that span the year with baseline start and end dates, an estimate of the percent annual budget and a high-level description. Eight weeks before the start of an activity, full details about staffing, completion criteria, and more are added to the issue. Figure 17 shows the steps in diagram form.

Cost, scope, and schedule for the ECP ST are tracked and managed by monitoring the progress of the collection of P6 Activities. Value is accrued when a P6 Activity issue is marked “Done” in the Jira database. Schedule and cost performance indices are derived from the status of the P6 Activities. Schedule, cost, and scope changes against the plan are managed via a formal project change request process.

2.2.2 KPP-3

The ECP has four KPPs (Table 3). KPP-3 focuses on a productive and sustainable software ecosystem. The ECP ST is the primary owner of this KPP along with co-design projects in the ECP AD. The focus of KPP-3 is to define and track capability integrations of ST products into the client environment, as described in this section.

KPP ID	Description of scope	Threshold KPP	Objective KPP	Verification action/evidence
KPP-1	Performance improvement for mission-critical problems	50% of selected applications achieve Figure of Merit improvement ≥ 50	100% of selected applications achieve their KPP-1 stretch goal	Independent assessment of measured results and report that threshold goal is met
KPP-2	Broaden the reach of exascale science and mission capability	50% of selected applications can execute their challenge problem	100% of selected applications can execute their challenge problem stretch goal	Independent assessment of mission application readiness
KPP-3	Productive and sustainable software ecosystem	50% of the weighted impact goals are met	100% of the weighted impact stretch goals are met	Independent assessment verifying threshold goal is met
KPP-4	Enrich the HPC hardware ecosystem	Vendors meet 80% of all the PathForward milestones	Vendors meet 100% of all the PathForward milestones	Independent review of the PathForward milestones to assure they meet the contract requirements; evidence is the final milestone deliverable

Table 3: The ECP’s four KPPs.

The following terms are defined for clarity.

- **Capability:** Any significant product functionality, including existing features adapted to the pre-exascale and exascale environments, that can be integrated into a client environment.
- **Integration goal:** A statement of impact on the ECP ecosystem in which a software capability is used in a consequential and sustainable way by a client in pre-exascale environments first, then in exascale environments. Integration goals are product focused. A project that contributes to more than one product will have a KPP-3 Jira issue for each of its products.
- **Integration score:** The number of successful capability integrations into a client environment (Table 4).
- **Sustainable:** For KPP-3, *sustainable* means that the capability is integrated in a way that reasonably ensures future use of the capability beyond the end of the ECP. For libraries, *sustainable* generally means that library usage is made from the source code in the main repository and use of the library is tested regularly as a part of the client code’s regular regression testing. For tools, *sustainable* generally means that the tool is available as needed in the exascale environment. For prototype capabilities that are incorporated into vendor and community software, the impact of the prototype is still visible to subject matter experts (SMEs).

Defining an Integration Goal Integration goals are defined per product within each project. The goal statement will include the following.

- The name of the product to which the project contributes. The product must be listed in the *ECP ST Product Dictionary*.
- A description of the target clients’ environments into which the product capabilities will be integrated. Specific clients can be listed but are not necessary. Clients must be part of the ECP or otherwise part of the exascale systems ecosystem, such as a vendor or facility partner.
- A general description of the nature of the integration that addresses what it means to be successfully integrated.

Demonstrating and Recording Progress toward an Integration Goal: All artifacts and evidence of progress will be captured in the Jira KPP-3 issue associated with a product integration goal as progress is made. All integration scores are tentative until the capability is available and demonstrated in exascale environments. Table 5 summarizes the defined values.

Integration score	Capability	Integration description
One point per capability sustainably integrated by a client per exascale platform used.	Complete, sustainable integration of a significant product capability into a client environment in a pre-exascale environment (tentative score) and in an exascale environment (confirmed score).	Clients acknowledge benefit from product capability use and considers it part of their workflow. Integration is sustainable with documentation and testing. Integration of product capability into main product repo and SDK/E4S environments is completed.

Table 4: Integration goal scoring. One point is accrued when a client integrates and sustainably uses a product’s capabilities. Scores are assessed annually.

Value	Definition	Description
Present	The current integration score.	This is always an indication of the progress that the team has made. The present value is assessed annually.
Passing	The minimum integration score required for the product to be counted as part of ECP ST progress toward KPP-3.	The passing score is between 4 and 8 for each integration goal, in which 4 is for larger integration efforts, and 8 is for smaller ones. This is equivalent to accomplishing one to two capability integrations per year, per product.
Stretch	The maximum reasonably achievable integration score for a product if capability integrations are successful with all potential ECP clients.	The stretch value shows the overall integration potential.

Table 5: Key metric values. These values are determined by the L4 subproject team when defining its KPP-3 issue.

Assessment Process Although progress is recorded as it is achieved, progress assessment is done annually, including input from external SMEs. ECP leadership and SMEs will review integration score evidence, confirming or adjusting integration scores. Assessments can result in a reduced integration score from a previous year if a client has stopped using a capability that was previously used.

Transition from Tentative to Confirmed Integration Score Each integration score is tentative until the capability is available and demonstrated to be effective in the exascale environments. Demonstration can be achieved in a variety of ways so that ECP leadership and SMEs are reasonably certain that the capability positively impacts the client in exascale environments. At this point, the integration score becomes confirmed. Typically, the transition from tentative to confirmed would be a low-cost, independent demonstration or accomplished within the client’s environment as the client is conducting its own assessments. The planned exascale system, El Capitan, that can support National Security Applications will not be available until the end of FY23. Integrating ST products into National Security Applications will be considered for transition from tentative to confirmed when: (1) evidence of integration is provided during FY20–22 ASC L1 and L2 milestones related to ECP/ATDM National Security Application readiness for exascale platforms and/or (2) integration is demonstrated on the El Capitan early access systems and exercises capabilities similar to those anticipated to be important for effectively using El Capitan. For KPP-3 capability integrations targeted at El Capitan, the ECP will use the best available confirmation process in FY23. Table 6 explains the weight of the integration scores.

The KPP-3 score is the weighted sum of all integration goals that have an integration score that meets or exceeds its passing value. The KPP-3 score will initially be tentative. The KPP-3 score is not officially met until the weighted sum of confirmed integration scores exceeds 50% of the total possible points.

Impact level	Weight	Comments
High	2	The score for integration goals associated with high-impact products will be added to the KPP-3 score with a weight of 2.
Normal	1	Most KPP-3 Jira issues will have a weight of 1.
Risk mitigating	0.5	Some KPP-3 Jira issues are associated with products that help plan for the potential risks if high-impact products do not deliver as expected.
Shared	0.5	Some projects receive funding from both NNSA and DOE SC (e.g., RAJA/Kokkos). For these projects, the score is balanced to reflect dual contributions.

Table 6: Integration scores. Each integration score will have an associated weight, depending on the potential impact if integration targets are not met.

2.2.3 ECP ST Software Delivery

One essential activity for—and the ultimate purpose of—ECP ST is the delivery of a software stack that enables productive and sustainable exascale computing capabilities for target ECP applications and platforms and the broader HPC community. The ECP ST Software Ecosystem and Delivery sub-element (WBS 2.3.5), and the SDKs in each sub-element, provide the means by which the ECP ST will deliver its capabilities.

ECP ST Delivery and HI Deployment Providing the ECP ST software stack to ECP applications requires coordination between the ECP ST and ECP HI. The focus areas have a complementary arrangement in which the ECP ST delivers its products, and the ECP HI deploys them. The process is described specifically as follows.

- The ECP ST delivers software. ECP ST products are delivered directly to application teams, vendors, and facilities. The ECP ST designs and implements products to run on DOE computing facilities platforms and make products available as source code via GitHub, GitLab, or another accessible repository.
- The ECP HI facilitates efforts to deploy ST and other software on facilities platforms by installing them where users expect to find them. This could be in `/usr/local/bin` or a similar directory or available via “module load.”

Separating the concerns of delivery and deployment is essential because these activities require different skill sets. Furthermore, the ECP ST delivers its capabilities to an audience that is beyond the scope of specific facilities’ platforms. This broad scope is essential for the sustainability of ECP ST products, expanding the user and developer communities needed for vitality. Additionally, the ECP HI, computer system vendors, and other parties provide deployable software outside the scope of the ECP ST, thus having the critical skills needed to deploy the entire software stack.

ECP ST Delivery Strategy The ECP ST delivers its software products as source code, primarily in repositories found on GitHub or GitLab installations or similar platforms. Clients such as the ECP HI, OpenHPC, and application developers with direct repository access then take the source and build, install, and test the ECP ST software. The delivery strategy is outlined in Figure 18, and users access ECP ST products using the following basic mechanisms.

Build from Source Code Most ECP ST products reach at least some of their user base via direct source code download from the product repository. In some cases, users download one compressed file that contains the product source, then expand the file to expose the collection of source and build files. Increasingly, users will fork a new copy of an online repository. After obtaining the source, users execute a configuration process that detects local compilers and libraries and then builds the product. This kind of access can be a barrier for some users because users need to build the product and can encounter a variety of challenges in that process, such as an incompatible compiler or a missing third-party library that must first be installed. However, building from the source can be a preferred approach for users who want control over compiler settings or

Delivering an Open, Hierarchical Software Ecosystem

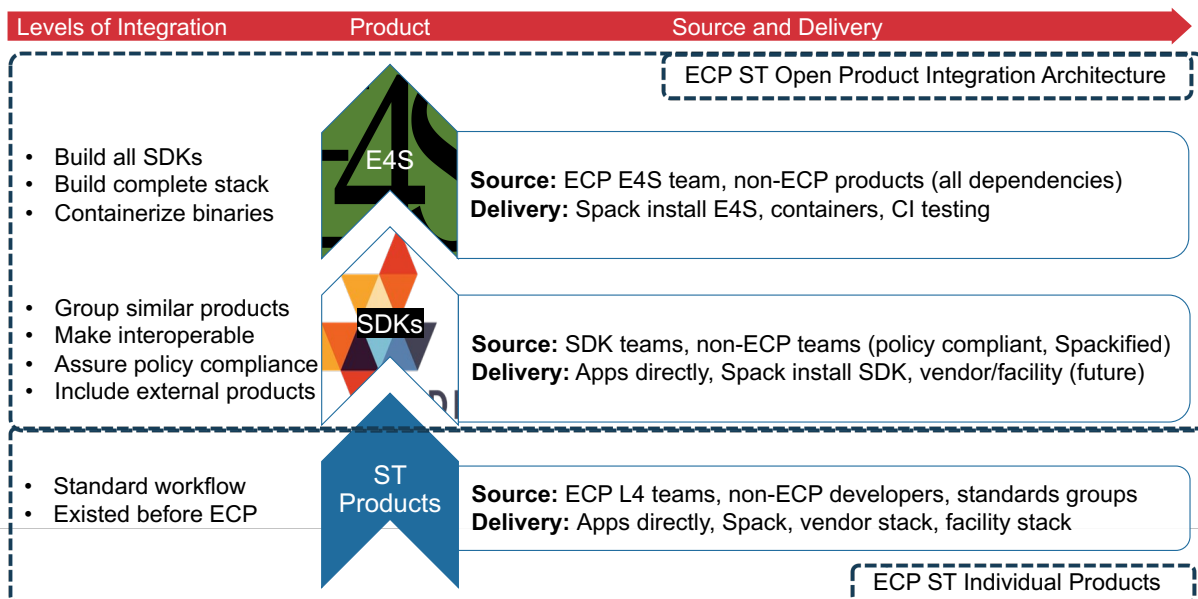


Figure 18: The ECP ST software stack is delivered to the user community through several channels, including via source code, using SDKs, direct to facilities in collaboration with ECP HI, and via binary distributions from containers and HPC vendors. Increasingly, E4S is the primary pathway for delivering ECP ST capabilities. E4S provides testing, a documentation portal, and quality commitments via community policies.

want to adapt how the product is used (e.g., turning on or off optional features, creating adaptations that extend product capabilities). For example, large library frameworks, such as PETSc and Trilinos, have many tunable features that can benefit from users building from source code. Furthermore, these frameworks support user-defined functional extensions that are easier to support when users build the product from source. The ECP ST is leveraging and contributing to the development of Spack [1]. Via metadata stored in a Spack package defined for each product, Spack leverages a product’s native build environment, along with knowledge about its dependencies, to build the product and dependencies from source. Spack plays a central role in ECP ST software development and delivery processes by supporting turnkey builds of the ECP ST software stack for the purposes of continuous integration testing, installation, and seamless multiproduct builds.

DOE Computing Facilities Each DOE computing facility—Argonne Leadership Computing Facility (ALCF), Oak Ridge Leadership Computing Facility (OLCF), National Energy Research Scientific Computing Center (NERSC), LLNL, and the Atmosphere, Climate, and Ecosystem Science (ACES) collaboration (Los Alamos National Laboratory [LANL] and Sandia National Laboratories [SNL]) provides pre-built versions of 17–20 ECP ST products, although the exact mix of products varies somewhat at each site. Many of these products are what users would consider to be part of the core system capabilities, including compilers—such as Flang (Section 4.2.27) and LLVM (Section 4.2.20)—and parallel programming environments, such as MPICH (Section 4.1.8), OpenMPI (Section 4.1.14), and OpenMP (Section 4.2.22). Development tools, such as PAPI (Section 4.2.8) and TAU (Section 4.2.17), are often part of this suite if they are not already included in the vendor stack. Math and data libraries, such as PETSc (Section 4.3.10), Trilinos (Section 4.3.17), HDF5 (Section 4.4.13), and others are also available in some facilities’ software installations. The team anticipates and hopes for increased collaboration with facilities via the ECP HI focus area. The team is also encouraged by multi-lab efforts, such as the Tri-Lab Operating System Stack (TOSS) [15], that are focused on improving the uniformity of software stacks across facilities.

Vendor Stacks Computer system vendors leverage DOE investments in compilers, tools, and libraries. Of particular note is the wide use of MPICH (Section 4.1.8) as a software base for most HPC vendor MPI implementations and the requirements, analysis, design, and prototyping that ECP ST teams provide. Section 3.3 describes some of these efforts.

Binary Distributions Approximately 10 ECP ST products are available via binary distributions, such as common Linux distributions, particularly via OpenHPC [16]. The ECP ST intends to foster the growth of availability via binary distributions as an important way to increase the size of the user community and improve product sustainability via this broader user base.

Container and Cloud Environments E4S is available via an increasing number of container and cloud environments, such as Docker, Shifter, Singularity, CharlieCloud, AWS, and Google Cloud.

2.2.4 ECP ST Software Life Cycle

This section discusses the ECP ST software life cycle, as shown in Figure 19. From their inception as P6 Activity planning packages, which are refined annually and given detailed information from before starting the activity all the way to the successful integration of a capability into the client environment, ECP ST features are governed by this life cycle. Each product team conducts its own integration planning that incorporates other funding sources and stakeholders, but the ECP ST life cycle intersects the product life cycle for capabilities that the ECP funds.

ECP ST Life Cycle Summary

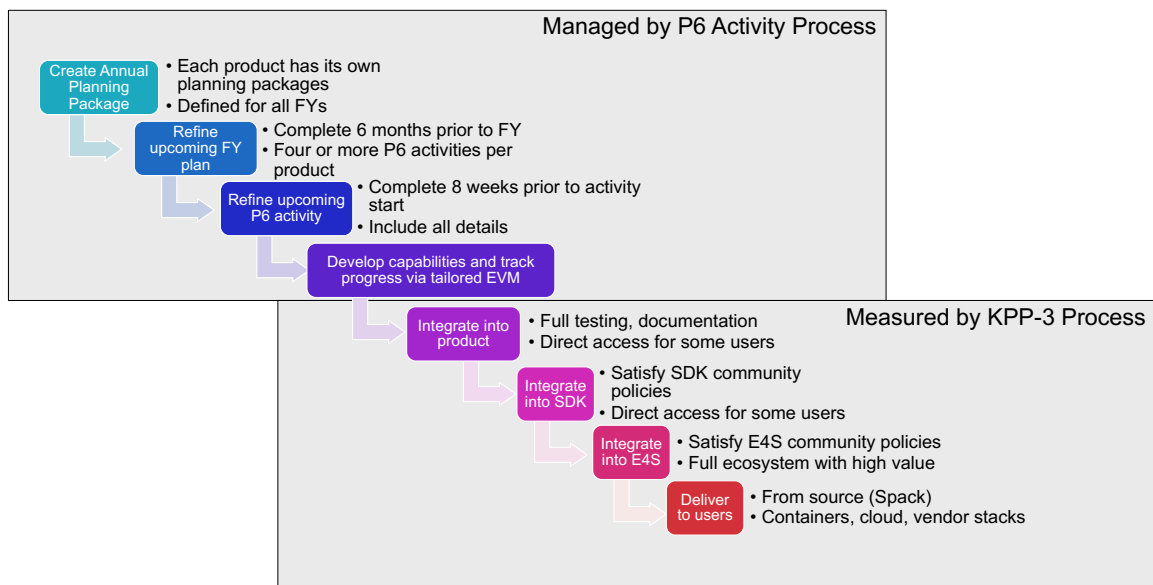


Figure 19: ECP ST product planning, execution, testing, and assessment are governed by the combination of P6 Activities for hierarchical planning (Figure 17) and KPP-3 for measuring capability integrations (Table 4). This figure shows how the entire life cycle of ECP ST feature development is captured by these two elements.

3. ECP ST DELIVERABLES

ECP ST efforts contribute to the HPC software ecosystem in a variety of ways. The most tangible way was to include contributions to software products, many of which are already widely deployed and being transformed for use with exascale systems. However, the ECP ST contributes to industry and de facto standards efforts. Finally, some ECP ST efforts contribute to the upstream processes of requirements, analysis, design, and prototyping that inform the implementation of vendor and other third-party software products. Although they do not receive the most attention, these upstream efforts are very impactful and low cost without a product to support.

ECP ST contributes to the HPC software ecosystem through direct product development, contributions to industry and de facto standards, and shaping the requirements, design, and prototyping of products delivery by vendors and other third parties.

3.1 ECP ST DEVELOPMENT PROJECTS

ECP ST efforts support development in the following software projects spanning five technical areas: (1) Programming Models & Runtimes, see Table 7; (2) Development Tools, see Table 8; (3) Mathematical Libraries, see Table 9; (4) Data & Visualization, see Table 10; and (5) Software Ecosystem & Delivery, see Table 11. Each table lists related projects, a URL (if available), and an estimate of deployment scope.

Product	Website	Deployment scope
GASNet-EX	https://gasnet.lbl.gov	Broad
Kokkos	https://github.com/kokkos	Broad
MPICH	http://www.mpich.org	Broad
OpenMPI	https://www.open-mpi.org	Broad
RAJA	https://github.com/LLNL/RAJA	Broad
CHAI	https://github.com/LLNL/CHAI	Moderate
Intel GEOPM	https://geopm.github.io	Moderate
Legion	http://legion.stanford.edu	Moderate
Qthreads	https://github.com/Qthreads	Moderate
Umpire	https://github.com/LLNL/Umpire	Moderate
UPC++	https://upcxx.lbl.gov	Moderate
UMap	https://github.com/LLNL/umap	Moderate
Variorum	https://github.com/LLNL/variorum	Moderate
BOLT	https://github.com/pmodels/bolt	Experimental
Argobots	https://github.com/pmodels/argobots	Experimental
PaRSEC	http://icl.utk.edu/parsec	Experimental
AML	https://xgitlab.cels.anl.gov/argo/aml	Experimental
PowerSlurm	https://github.com/tpatki/power-slurm	Experimental

Table 7: Programming Models & Runtimes (18 total).

Product	Website	Deployment scope
Caliper	https://github.com/llnl/caliper	Broad
Dyninst Binary Tools Suite	http://www.paradyn.org	Broad
Flang/LLVM Fortran compiler	http://www.flang-compiler.org	Broad
HPCToolkit	http://hpctoolkit.org	Broad
LLVM	http://llvm.org/	Broad
PAPI	http://icl.utk.edu/exa-papi	Broad
SCR	https://github.com/llnl/scr	Broad
STAT	https://github.com/LLNL/STAT	Broad
TAU	http://tau.uoregon.edu	Broad
LLVM OpenMP compiler	https://github.com/SOLLVE	Moderate
OpenMP V & V Suite	https://bitbucket.org/crpl_cisc/sollve_vv/src	Moderate
mpiFileUtils	https://github.com/hpc/mpifileutils	Moderate
openarc	https://ft.ornl.gov/research/openarc	Moderate
Papyrus	https://csmd.ornl.gov/project/papyrus	Moderate
Program DB Toolkit	https://www.cs.uoregon.edu/research/pdt	Moderate
PRUNERS Toolset	https://github.com/PRUNERS/PRUNERS-Toolset	Moderate
TriBITS	https://tribits.org	Moderate
Gotcha	http://github.com/llnl/gotcha	Experimental
Kitsune	https://github.com/lanl/kitsune	Experimental
Metall	https://github.com/LLNL/metall	Experimental
QUO	https://github.com/lanl/libquo	Experimental
SICM	https://github.com/lanl/sicm	Experimental
SuRF		Experimental

Table 8: Development Tools (22 total).

Product	Website	Deployment scope
hypr	http://www.llnl.gov/casc/hypr	Broad
Kokkoskernels	https://github.com/kokkos/kokkos-kernels	Broad
MFEM	http://mfem.org/	Broad
PETSc/TAO	http://www.mcs.anl.gov/petsc	Broad
SLATE	http://icl.utk.edu/slate	Broad
SUNDIALS	https://computing.llnl.gov/sundials	Broad
SuperLU	https://portal.nersc.gov/project/sparse/superlu	Broad
Trilinos	https://github.com/trilinos/Trilinos	Broad
DTK	https://github.com/ORNL-CEES/DataTransferKit	Moderate
FleCSI	http://www.flecsi.org	Moderate
MAGMA-sparse	https://bitbucket.org/icl/magma	Moderate
STRUMPACK	http://portal.nersc.gov/project/sparse/strumpack	Moderate
xSDK	https://xsdk.info	Moderate
FFTX	https://github.com/spiralgen/fftx	Experimental
ForTrilinos	https://trilinos.github.io/ForTrilinos	Experimental
libEnsemble	https://github.com/Libensemble/libensemble	Experimental
Tasmanian	http://tasmanian.ornl.gov	Experimental
ArborX	https://github.com/arborx/ArborX	Experimental

Table 9: Mathematical Libraries (18 total).

Product	Website	Deployment scope
Catalyst (ALPINE)	https://www.paraview.org/in-situ	Broad
Darshan	http://www.mcs.anl.gov/research/projects/darshan	Broad
HDF5	https://www.hdfgroup.org/downloads	Broad
IOSS	https://github.com/gsjardema/seacas	Broad
Parallel netCDF	http://cucis.ece.northwestern.edu/projects/PnetCDF	Broad
ParaView (ALPINE)	https://www.paraview.org	Broad
ROMIO	http://www.mcs.anl.gov/projects/romio	Broad
VeloC	https://veloc.readthedocs.io	Broad
VisIt (ALPINE)	https://wci.llnl.gov/simulation/computer-codes/visit	Broad
VTK-m	http://m.vtk.org	Broad
ADIOS	https://github.com/ornladios/ADIOS2	Moderate
ASCENT (ALPINE)	https://github.com/Alpine-DAV/ascent	Moderate
In Situ Algorithms (ALPINE)	https://github.com/Alpine-DAV/algorithms	Moderate
Cinema	https://github.com/cinemascience	Moderate
zfp	https://github.com/LLNL/zfp	Moderate
SZ	https://szcompressor.org	Moderate
C2C		Experimental
FAODEL	https://github.com/faodel/faodel	Experimental
GUFI	https://github.com/mar-file-system/GUFI	Experimental
HXHIM	http://github.com/hpc/hxhim.git	Experimental
MarFS	https://github.com/mar-file-system/marfs	Experimental
Mercury	http://www.mcs.anl.gov/research/projects/mochi	Experimental
ROVER	https://github.com/LLNL/rover	Experimental
Siboka		Experimental
UnifyFS	https://github.com/LLNL/UnifyFS	Experimental

Table 10: Data & Visualization (25 total).

Product	Website	Deployment scope
Spack	https://github.com/spack/spack	Broad
E4S	https://e4s.io	Broad

Table 11: Software Ecosystem & Delivery (two total).

3.2 STANDARDS COMMITTEES

Participating in standards efforts is an important activity for ECP ST staff. In many instances, the software will not be sustainable if it is not tightly connected to a standard. Additionally, any standard must account for the emerging requirements that exascale platforms need to meet to achieve performance and portability. Table 12 summarizes ECP ST staff involvement in the major standards efforts that impact the ECP.

ECP ST staff are heavily involved in the MPI and OpenMP standards efforts and hold several key leadership positions within the standardizing bodies. ECP ST staff also play a critical role in C++ standards efforts. Although DOE staff have only recently engaged in C++ standards, ECP ST staff efforts are essential for ensuring that HPC requirements are considered, especially by contributing working code that demonstrates requirements and design. The ECP ST sponsors the newest open-source Fortran compiler, Flang 4.2.27 as a front end for LLVM. This compiler is rapidly emerging as an essential part of the HPC ecosystem. In particular, although ARM processors are not explicitly part of the pre-exascale ecosystem, they are

emerging as a strong contender in the future. Flang is *the* Fortran compiler for ARM-based systems. ECP ST involvement in other committees provides valuable leverage and improved uniformity for HPC software.

Standards effort	ECP ST participants
MPI Forum	15
OpenMP	15
BLAS	6
C++	4
Fortran	4
OpenACC	3
LLVM	2
PowerAPI	1
VTK ARB	1

Table 12: ECP ST staff are involved in a variety of official and de facto standards committees. Involvement in standards efforts is essential for ensuring the sustainability of ECT ST products and ensuring that emerging exascale requirements are addressed by these standards.

3.3 CONTRIBUTIONS TO EXTERNAL SOFTWARE PRODUCTS

Although many ECP ST efforts focus on the product that it will develop and support, some of the important work—and certainly some of the most sustainable and highly leveraged work—is done by providing requirements, analysis, design, and prototype capabilities to vendor and other third-party software. Many software studies have shown that 70%–80% of the cost of a successful software product goes into post-delivery maintenance. The effort summarized in Table 13 expressly eliminates this large cost for DOE because the product is developed and supported outside DOE.

Product	Contribution
Kokkos and RAJA	ECP efforts to provide portable on-node parallel programming and execution environments have led to new features in C++ standards.
MPI Forum	ECP ST staff maintain several chapters of the MPI Forum, an effort that requires constant involvement with the other authors, as well as participation in the online discussions related to the chapter and regular attendance of the MPI Forum’s face-to-face activities.
Flang	ECP funds the development of the new open-source Fortran compiler front end called Flang. Flang provides Fortran language support for LLVM back ends in a way similar to how Clang provides support for C and C++.
All Development Tools work	Starting in FY20, the Development Tools efforts are organized around delivering capabilities into the LLVM ecosystem.
SWIG	The ECP ST ForTrilinos efforts contribute the capability to generate automatic Fortran bindings from C++ code.
TotalView debugger	ECP ST staff, along with RogueWave engineers, are engaged in the co-design of OMPD, the new debugging interface for OpenMP programs. This effort helps RogueWave improve its main debugging product, TotalView, by making it aware and compatible with recent advances in OpenMP debugging.
LLVM	An ECP ST staff member is co-leading design discussions around the parallel intermediate representation (IR) and loop-optimization infrastructure.
SLATE	ECP ST Mathematical Library efforts inform the design, implementation, and optimization of dense numerical linear algebra routines on most vendor platforms.
Cray MPICH MPI-IO	As part of the ExaHDF5 ECP project, the ALCF worked with Cray MPI-IO developers to merge the upstream ROMIO code into the downstream proprietary Cray MPICH MPI-IO, leveraging Cray’s extensive suite of I/O performance tests and further tuning the algorithm. Cray is currently targeting its deployment in an experimental release.
OpenHPC	An ECP ST staff member serves on the OpenHPC Technical Steering Committee as a component development representative.

Table 13: External products to which ECP ST activities contribute. Participation in requirements, analysis, design, and prototyping activities for third-party products is some of the most effective software work that can be done.

4. ECP ST TECHNICAL AREAS

ECP ST is composed of six L3 technical areas (Figure 20). This section of the report provides an overview of each L3 technical area and two-page summaries of each funded project within the technical area. For each L3 area, the report discusses scope and requirements, assumptions and feasibility, objectives, plans, risks and mitigations, and future trends. For each L4 subproject, a project overview is provided and the key challenges, solution strategy, recent progress, and next steps for the project are summarized.

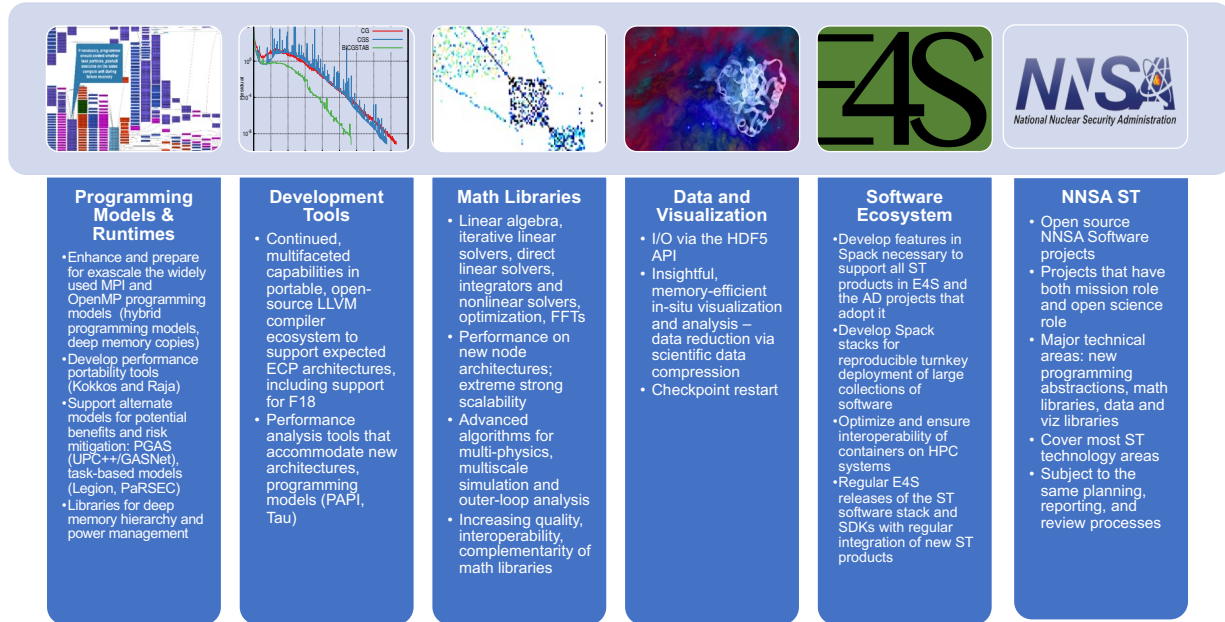


Figure 20: ECP ST is composed of six L3 technical areas. The first four areas are organized around functionality development themes. The fifth is focused on technology for packaging and delivering capabilities. The sixth is organized around per-lab open-source development at the three DOE NNSA laboratories: LANL, LLNL, and Sandia.

4.1 WBS 2.3.1 PROGRAMMING MODELS & RUNTIMES

End State: A cross-platform, production-ready programming environment that enables and accelerates the development of mission-critical software at both the node and full-system levels.

4.1.1 Scope and Requirements

A programming model provides the abstract design upon which developers express and coordinate the efficient parallel execution of their program. A particular model is implemented as a developer-facing interface and a supporting set of runtime layers. To successfully address the challenges of exascale computing, these software capabilities must address the challenges of programming at both the node- and full-system levels. These two targets must be coupled to support multiple complexities expected with exascale systems (e.g., locality for deep memory hierarchies, affinity for threads of execution, load balancing) and also provide a set of mechanisms for performance portability across the range of potential and final system designs. Additionally, there must be mechanisms for the interoperability and composition of multiple implementations (e.g., one at the system level and one at the node level). This must include abilities such as resource sharing for workloads that include coupled applications, supporting libraries and frameworks, and capabilities such as in situ analysis and visualization.

Given the ECP’s timeline, the development of new programming languages and their supporting infrastructure is infeasible. We do, however, recognize that the augmentation or extension of the features of existing and widely used languages (e.g., C/C++ and Fortran) could provide solutions for simplifying certain software development activities.

4.1.2 Assumptions and Feasibility

The intent of the PMR L3 is to provide a set of programming abstractions and their supporting implementations that allow programmers to select from options that meet demands for expressiveness, performance, productivity, compatibility, and portability. It is important to note that, while these goals are obviously desirable, they must be balanced with an additional awareness that today’s methods and techniques may require changes in both the application and the overall programming environment and within the supporting software stack.

4.1.3 Objectives

PMR provides the software infrastructure necessary to enable and accelerate the development of HPC applications that perform well and are correct and robust, while reducing the cost both for initial development and ongoing porting and maintenance. PMR activities need to reflect the requirements of increasingly complex application scenarios, usage models, and workflows, while at the same time addressing the hardware challenges of increased levels of concurrency, data locality, power, and resilience. The software environment will support programming at multiple levels of abstraction that includes both mainstream as well as alternative approaches if feasible in ECP’s timeframe.

Both of these approaches must provide a portability path such that a single application code can run well on multiple types of systems, or multiple generations of systems, with minimal changes. The layers of the system and programming environment implementation will therefore aim to hide the differences through compilers, runtime systems, messaging standards, shared-memory standards, and programming abstractions designed to help developers map algorithms onto the underlying hardware and schedule data motion and computation with increased automation.

4.1.4 Plan

PMR contains nine L4 projects. To ensure relevance to DOE missions, these efforts leverage and collaborate with existing activities within the broader HPC community. The PMR area supports the research and development needed to produce exascale-ready versions of the Message Passing Interface (MPI); Partitioned Global-Address Space Libraries (UPC++, GASNet); task-based programming models (Legion, PaRSEC); software for node-level performance portability (Kokkos, RAJA); and libraries for memory, power, and resource management. Initial efforts focused on identifying the core capabilities needed by the selected ECP applications and components of the software stack, identifying shortcomings of current approaches, establishing performance baselines of existing implementations on available petascale and prototype systems, and the re-implementation of the lower-level capabilities of relevant libraries and frameworks. These efforts provided demonstrations of parallel performance of algorithms on pre-exascale, leadership-class machines—at first on test problems, but eventually in actual applications (in close collaboration with the AD and HI teams). Initial efforts also informed research into exascale-specific algorithms and requirements that will be implemented across the software stack. The supported projects targeted and implemented early versions of their software on CORAL, NERSC and ACES pre-exascale systems—with an ultimate target of production-ready deployment on the exascale systems. In FY20–23, the focus is on development and tuning for the specific architectures of the selected exascale platforms, in addition to tuning specific features that are critical to ECP applications.

Throughout the effort, the applications teams and other elements of the software stack evaluate and provide feedback on their functionality, performance, and robustness. Progress towards these goals is documented quarterly and evaluated annually (or more frequently if needed) based on PMR-centric milestones as well as joint milestone activities shared across associated software stack activities by Application Development and Hardware & Integration focus areas.

4.1.5 *Risks and Mitigation Strategies*

The mainstream activities of ECP in the area of programming models focus on advancing the capabilities of MPI and OpenMP. Pushing them as far as possible into the exascale era is key to supporting an evolutionary path for applications. This is the primary risk mitigation approach for existing application codes. Extensions to MPI and OpenMP standards require research, and part of the efforts will focus on rolling these findings into existing standards, which takes time. To further address risks, PMR is exploring alternative approaches to mitigate the impact of potential limitations of the MPI and OpenMP programming models.

Another risk is the failure of adoption of the software stack by the vendors, which is mitigated by the specific delivery focus in sub-element SW Ecosystem and Delivery. Past experience has shown that a combination of laboratory-supported open-source software and vendor-optimized solutions built around standard APIs that encourage innovation across multiple platforms is a viable approach and is what we are doing in PMR. We are using close interaction with the vendors early on to encourage adoption of the software stack, including well-tested practices of including support for key software products or APIs into large procurements through NRE or other contractual obligations. A mitigation strategy for this approach involves building a long-lasting open-source community around projects that are supported via laboratory and university funding.

Creating a coordinated set of software requires strong management to ensure that duplication of effort is minimized. This is recognized by ECP management, and processes are in place to ensure collaboration is effective, shortcuts are avoided unless necessary, and an agile approach to development is instituted to prevent prototypes moving directly to product.

4.1.6 *Future Trends*

Recently announced exascale system procurements have shown that the trend in exascale compute-node hardware is toward heterogeneity: Compute nodes of future systems will have a combination of regular CPUs and accelerators (typically GPUs). Furthermore, the GPUs will not be just from NVIDIA as on existing systems: One system will have Intel GPUs and another will have AMD GPUs. In other words, there will be a diversity of GPU architectures, each with their own vendor-preferred way of programming the GPUs. An additional complication is that although the HPC community has some experience in using NVIDIA GPUs and the associated CUDA programming model, the community has relatively little experience in programming Intel or AMD GPUs. These issues lead to challenges for application and software teams in developing exascale software that is both portable and high performance. Below we outline trends in programming these complex systems that will help alleviate some of these challenges.

Trends in Internode Programming The presence of accelerator hardware on compute nodes has resulted in individual compute nodes becoming very powerful. As a result, millions of compute nodes are no longer needed to build an exascale system. This trend results in a lower burden on the programming system used for internode communication. It is widely expected that MPI will continue to serve the purpose of internode communication on exascale systems and is the least disruptive path for applications, most of which already use MPI. Nonetheless, improvements are needed in the MPI Standard as well as in MPI implementations in areas such as hybrid programming (integration with GPUs and GPU memory, integration with the intranode programming model), overall resilience and robustness, scalability, low-latency communication, optimized collective algorithms, optimized support for exascale interconnects and lower-level communication paradigms such as OFI and UCX, and scalable process startup and management. PGAS models, such as UPC++ and OpenSHMEM, are also available to be used by applications that rely on them and face similar challenges as MPI on exascale systems. These challenges are being tackled by the MPI and UPC++/GASNet projects in the PMR area.

Trends in Intranode Programming The main challenge for exascale is in achieving performance and portability for intranode programming, for which a variety of options exist. Vendor-supported options include CUDA and OpenACC for NVIDIA GPUs, SYCL/DPC++ for Intel GPUs, and HIP for AMD GPUs. OpenACC supports accelerator programming via compiler directives. SYCL provides a C++ abstraction on top of OpenCL, which itself is a portable, lower-level API for programming heterogeneous devices. Intel's

DPC++ is similar to SYCL with some extensions. HIP from AMD is similar to CUDA; in fact, AMD provides translation tools to convert CUDA programs to HIP.

Among portable, standard programming models, OpenMP has supported accelerators via the `target` directive starting with OpenMP version 4.0 released in July 2013. Subsequent releases of OpenMP (version 4.5 and 5.0) have further improved support for accelerators. OpenMP is supported by vendors on all platforms and, in theory, could serve as a portable intranode programming model for systems with accelerators. However, in practice, a lot depends on the quality of the implementation.

Kokkos and RAJA provide another alternative for portable, heterogenous-node programming via C++ abstractions. They are designed to work on complex node architectures with multiple types of execution resources and multilevel memory hierarchies. Many ECP applications are successfully using Kokkos and RAJA to write portable parallel code that runs efficiently on GPUs.

We believe these options (and high-quality implementations of them) will meet the needs of applications in the exascale timeframe.

4.1.7 WBS 2.3.1.01 Programming Models & Runtimes Software Development Kits

Overview The Programming Models & Runtimes SDK effort is focused on identifying meaningful aggregations of products in this technical area. SDK efforts are in the early stages of planning and execution. Most of the work on SDKs has been driven from the Software Ecosystem & Delivery technical area. A description of the SDK effort can be found in Section 4.5.7.

4.1.8 WBS 2.3.1.07 Exascale MPI

Overview MPI has been the de facto standard programming model for HPC from the mid 90's till today, a period where supercomputing performance increased by six orders of magnitude. The vast majority of DOE's parallel scientific applications running on the largest HPC systems use MPI. These application codes represent billions of dollars of investment. Therefore, MPI must evolve to run as efficiently as possible on Exascale systems. Our group at Argonne developed a high-performance, production-quality MPI implementation, called MPICH. The focus areas of the Exascale MPI/MPICH project are: (1) continuous improvement of the performance and capabilities of the MPICH software to meet the demands of ECP and other broader DOE applications, (2) coordinate vendor and supercomputing center interactions to ensure efficient solutions to applications, and (3) be involved in the MPI Forum and standardization efforts to ensure continuity of the work beyond this project.

The MPICH group was involved in the formation of the MPI Forum and has been deeply involved in defining the MPI standard since 1992. MPICH has helped prototype and define the majority of the features in the MPI standard. As such, MPICH has been one of the most influential pieces of software in accelerating the adoption of the MPI standard by the HPC community. MPICH has been adopted by leading vendors into their own derivative implementations. Examples include Intel (for Intel MPI), Cray (for Cray MPI), IBM (for IBM PE MPI), Mellanox (for MLNX-MPI), Microsoft (for MS-MPI), and Ohio State University (for MVA-PICH). MPICH and its derivatives are routinely used in the majority of the top 10 supercomputers in the world. MPICH is the recipient of a number of awards including an R&D 100 award.

Key Challenges While we believe MPI is a viable programming model at Exascale, both the MPI standard and MPI implementations have to address the challenges posed by the increased scale, performance characteristics and evolving architectural features expected in Exascale systems, as well as the capabilities and requirements of applications targeted at these systems. The key challenges are as follows:

1. Interoperability with intranode programming models having a high thread count [17, 18, 19] (such as OpenMP, OpenACC and emerging asynchronous task models);
2. Scalability and performance over complex architectures [19, 20, 21, 22] (including high core counts, processor heterogeneity and heterogeneous memory);
3. Software overheads that are exacerbated by lightweight cores and low-latency networks;

4. Enhanced functionality (extensions to the MPI standard) based on experience with applications and high-level libraries/frameworks targeted at Exascale; and
5. Topics that become more significant as we move to the next generation of HPC architectures: memory usage, power, and resilience.

Solution Strategy The Exascale MPI project has the following primary technical thrusts: (1) Performance and Scalability, (2) Heterogeneity, (3) Topology Awareness, (4) Fault Tolerance, and (5) MPI+X Hybrid Programming.

Our solution strategy started by addressing performance and scalability aspects in MPICH related to network address management [23]. Apart from this, we also looked at communication strategies which allow the MPI library to be as lightweight as possible [24, 25]. Other solutions include investigation and evaluation of communication relaxation hints, investigation of optimizations to memory scalability in MPICH and improvements to MPI RMA operations.

Exascale MPI heterogeneity efforts [26, 27, 28] started with the survey on heterogeneous memory architectures on upcoming DOE machines and how MPICH can take advantage of them [29]. The efforts also included the investigation of utilizing heterogeneous memory inside the MPI implementation and evaluation of applications [30]. The heterogeneity efforts further extended to investigating and developing technologies for GPU integration for the better support of the coming Exascale supercomputers.

Exascale MPI topology awareness efforts [31, 32] originated with the investigation and evaluation of hints based on topology awareness and optimizations to virtual topology functionality in MPICH [33, 34]. The other efforts include investigation of topology-aware collectives and neighborhood collectives in MPICH [35] and evaluation of the selected ECP applications.

Exascale MPI fault tolerance efforts [36, 19] started with support for handling noncatastrophic errors in MPI. The second effort included defining the scope of errors in MPI, a prerequisite for user-level failure mitigation. Other efforts in this direction include standardizing these efforts in MPI and evaluating application suitability for fault tolerance.

Exascale MPI+X hybrid programming efforts involved improving interoperation of MPICH with threads [37]. Secondly, we developed the work-queue data transfer model for multithreaded MPI communication [38]. We have included support for interaction of MPICH with user-level thread (ULT) libraries [39], primarily targeting Argobots and the BOLT runtime [40]. Other issues being investigated include the design and evaluation of multiple virtual communication interfaces (VCIs) for multithreaded MPI communication [41].

Recent Progress One major achievement of MPICH recently is the support for the newly released MPI-4 standard. The MPI-4 standard introduced new ways for communication including persistent collective that aims to reduce the overhead of repetitively setting up collective, and partitioned communication which aims to improve the performance for multi-threaded MPI communication. The MPICH 4.0 release has include all the new MPI-4 features.

We focus on better GPU support in MPICH. MPICH now supports GPUs from all three vendors (NVIDIA, AMD and Intel). The support for GPU communication includes a fast path that utilizes the hardware RDMA support for GPU memory, and a fallback path for cases like complex datatypes which cannot be directly handed by hardware RDMA. We have also developed prototypes for GPU-stream-triggered operations, which allow an MPI application to enqueue MPI point-to-point communication to a GPU stream or command queue. This allows the GPU stream to be used to manage the synchronization between GPU kernels and the MPI communication that depends on the result of those kernels. The benefit includes better overlapping between computation and communication, and higher GPU utilization.

Testing, Continuous Integration, and Experience on Early Access Systems Validation and verification has been an integral part of the MPICH project. We have made several improvements to the automated testing infrastructure for more effective testing. Specifically, we added the support for testing with GPU buffers for all three GPU vendors. It enables us to test MPI communications with different combinations of CPU and GPU buffer including inter-process communication between multiple GPUs on the same node. We regularly test MPICH using both Spock and Arcticus to evaluation and verify the new features and optimizations that targets the upcoming exascale systems. We have also setup a Continuous Integration (CI)

configuration to automate part of the test. These testing and verification helped us identify problems early and provides the necessary environment for experiments. It also simplifies our collaborations with vendor partners that we have similar hardware setup for testing on issues of common interests.

Next Steps In the next year, we plan to continue optimizing the GPU support in MPICH. Planned work includes performance evaluation and improvement of GPU communication, support for GPU collectives, and extending GPU-stream-triggered operations to more MPI communication functions. We will also expand the software validation and verification work on pre-exascale and exascale systems to make MPICH perform well on the target machines. We will also continue our close collaboration with the vendors, Intel and Cray, to ensure that high-quality MPICH-based MPI implementations (Intel MPI and Cray MPI) are available on the exascale platforms.

4.1.9 WBS 2.3.1.08 Legion

Overview This project focuses on the development, hardening and support of the Legion Programming System⁴ with a focus on Exascale platforms and workloads that can benefit from Legion’s features and capabilities. At a fundamental level, the project focuses s on the key capabilities (e.g. correctness, performance, scalability) of an alternative programming model for ECP that seeks to expose additional levels of parallelism and provide more flexible data-centric abstractions. In addition, Legion also enables a separation of concerns of the implementation of an application from how it is mapped onto a given system architecture.

Our efforts have been focused on addressing bugs, refactoring the implementation for improved stability, performance and scaling, extending support for the selected exascale platforms, and also expanding the feature set as needed for both application and platform requirements.

Key Challenges As with all new approaches to programming high-performance systems, there is a distinct challenge and a significant level-of-effort required to reach a level of stability, capabilities, and performance to establish stability, acceptance, and an established user base. In this regard, Legion is becoming more widely used outside of ECP for both machine-learning and data-centric workloads within industry (e.g., NVIDIA and Facebook). This growing base is providing both additional developers and use cases that are helping to stabilize the code as well as broaden the overall scope of testing. Much like these efforts benefit from ECP Legion-centric efforts, ECP is benefiting from this growth and additional external investments.

We have focused much of our efforts on emerging use cases within ECP that are a mix of traditional workloads, and those related to machine learning (ML) and data-centric computation. The data-centric and ML domains have provided a path for more substantial impact given reliance on external tools (e.g. TensorFlow, Python, etc.) vs. years of established code written in MPI. We continue to see clear benefits from focusing our efforts in these areas. This has helped us to increase our overall impact as well focus on areas of adoption across more specialized application needs that seeks to leverage machine learning and related workloads.

The ML component is support primarily via the FlexFlow framework that is discussed in more detail below. FlexFlow represents an additional level of scope that require its own set of efforts for hardening, expanding the feature set, tuning performance, and porting to the ECP target platforms. None of the FlexFlow scope was originally planned at the outset of ECP.

Solution Strategy As part of a larger collaboration between LANL, Stanford University, NVIDIA, SLAC, Facebook, MIT, and others this project provides the overarching implementation of Legion that captures the most stable (correct, feature complete, and performant) versions of the programming system for ECP’s use. In addition, we are actively looking for opportunities to further educate the broader community about Legion and general advantages of using data-centric and task-based approaches to programming.

We have continued working with Ristra (AD 2.2.5.01), ExaFEL (AD 2.2.4.05) and the CANDLE project (AD 2.2.4.03) to provide support for Legion. For all these efforts we provide support, software releases, and bug fixes related to both correctness and performance. Our project includes management of the current repository and quarterly, or more frequent, releases of Legion to the broader community (e.g. NVIDIA,

⁴<https://legion.stanford.edu>

Facebook, FlexFlow, etc.). In addition, we have provided initial support for AMD and Intel GPUs, and features that support inter-operation with other languages and programming systems – e.g. MPI, OpenMP, Kokkos, Fortran, and Python. Finally, we have collaborated at the lower levels of the software stack with the GASNet-EX effort (part of the Pagoda project in ST).

As part of the focused work with CANDLE, we are providing additional support with the implementation of the FlexFlow deep-learning framework that is built on top of Legion. This framework has proven to be significantly faster than industry-provided solutions such as TensorFlow and PyTorch – providing over a 15x reduction in training time on some of CANDLE’s networks.

Recent Progress We continue to discover and address both performance and scalability issues in the runtime. In addition, for use cases within ECP, and also a growing set of users outside of ECP, we have continued to identify and address bugs and other issues (e.g. missing features). The vast majority of these fixes have been released and are part of the Legion releases in use within ECP.

Additional work, some done in collaboration with the GASNet-EX team, has provided a Legion implementation that supports AMD’s GPU architectures as well as GPU-to-GPU network-based communications. In addition, the Kokkos interoperability support in Legion now also supports AMD’s GPU software stack and testing is underway with the Ristra team for full support on early access test-beds for AMD-based exascale platforms. Performance debugging will be an upcoming activity once appropriate resources are available.

Additional work has been done to support Intel’s GPUs and all of Legion’s internal CI tests and small test applications are successfully and correctly running on early access systems. Performance debugging will be an upcoming activity once appropriate resources are available.

We have continued to develop high-level entry points for FlexFlow that allow it to seamlessly use code from existing learning frameworks (e.g., Keras, PyTorch, and ONNX). We currently have versions of Keras/TensorFlow networks running correctly and are continuing to make progress on support for PyTorch and ONNX. Additional work has been done to improve performance via exposing new levels of parallelism in training networks as well as graph-centric optimizations. Our latest results suggest a 2-3x performance boost over previous results for one of CANDLE’s example learning networks. Like Legion, we currently use a quarterly set of releases for FlexFlow that are aligned to leverage features and new capabilities of Legion and the underlying software components on the target platforms.

Both FlexFlow and Legion are open source projects and distributed via GitHub:

- Legion: <https://github.com/StanfordLegion/legion/>
- FlexFlow: <https://github.com/flexflow/FlexFlow>

Next Steps Our plans for the next year are to continue focusing on the challenges presented by obtaining reliable, high performance versions of Legion and FlexFlow on the upcoming exascale system architectures.

We will continue to work on the Python interfaces for Legion (an aspect of growing interest from the ExaFEL project for improved productivity) and the feature set of FlexFlow requested by CANDLE. This will include seeking out and improving our outreach and leveraging new features of Legion exposed by other non-ECP efforts. Regular open-source releases of Legion and FlexFlow will continue, and as we test on the early access systems we will continue to focus on bug fixes, improving capabilities and developer productivity, and addressing performance issues.

4.1.10 WBS 2.3.1.09 Distributed Tasking at Exascale: PaRSEC

Overview The PaRSEC Environment (Figure 21) provides a software ecosystem composed of a runtime component to dynamically execute task-based applications on heterogeneous distributed systems, and a productivity toolbox that comprises a development framework for the support of multiple domain-specific languages (DSLs) and extensions, with debugging, trace collection, and analysis tools.

The PaRSEC project team is dedicated to solving two challenging and interdependent problems facing the ECP developer community: First, how to create an execution model that enables developers to express as much parallelism as possible in their applications, so that applications effectively utilize the massive collection of heterogeneous devices ECP machines will deploy. Second, how to ensure the execution model is flexible and

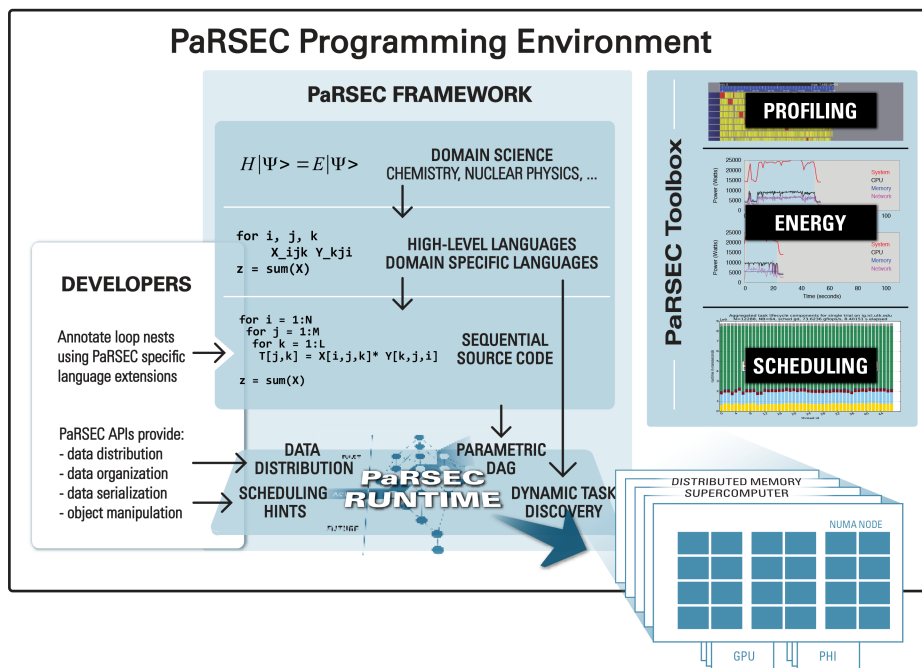


Figure 21: PaRSEC modular framework allows each component to be dynamically activated.

portable enough to actually provide and sustain a performance benefit by increasing the scientific productivity of the application developers, not only for the ECP target environments but for the foreseeable future.

PaRSEC is an open source, community-based implementation of a generic task-based runtime that is freely available, and used by an increasing number of software libraries. The project focuses on providing a stable and efficient infrastructure for quick prototyping of different approaches to define task-based languages able to exploit the full range of capabilities of Exascale platforms. Without such a project, and based on the current state of task-based runtime, potential users will be stuck either in fixed programming paradigms, or with a particular, potentially less efficient, mix of programming languages. The DTE project provides means to maintain a high competitiveness in the field leading to more innovation on addressing the challenges we are facing toward scalable, efficient and Exascale-ready programming paradigms.

Key Challenges As we get closer to the delivery of Exascale platforms, an increasing number of aspects of the hardware and software environment still pose challenges. First and foremost, keeping pace with the architectural changes on current and future platforms requires changes not only on how we take advantage of the hardware capabilities, but how we reshape our algorithms and applications to expose enough parallelism to maximize the use of the underlying hardware. The number of nodes, threads per node, memory hierarchies and support for increased computational capabilities (accelerators) will continue to increase, while the currently available programming paradigms are still struggling with parallelism at the node level.

The approach followed in PaRSEC is to provide a low-level, flexible and dynamic runtime able not only to schedule tasks at the node level, but to handle data transfers between different memory (both inter- and intra-nodes), memory hierarchies, heterogeneous architectures with support for accelerators with a simple programming scheme. The proposed approach envisions a middle-ground solution, addressing both hardware and software challenges. At the hardware level a team of dedicated developers extends PaRSEC to map its capabilities to the hardware and to improve its scalability and performance. At the upper software level the runtime interactions are through Domain Specific Languages with the target domain scientists in mind, that will facilitate the expression of algorithmic parallelism with familiar constructs mapped on the exposed low-level capabilities. To facilitate the integration of PaRSEC-driven libraries into larger and complex applications, PaRSEC natively interoperate with other programming paradigms, including some

target of the ECP PMR support, such as PGAS, MPI, OpenMP and Kokkos. This integration provides a smooth transition for library developers that embrace the PaRSEC runtime, providing a platform where a shift to a new programming paradigms can be done in stages of increased complexity [42, 43, 44].

The PaRSEC team, in collaboration with NWChemEx project researchers, developed an efficient and portable tensor product algorithm specifically designed for the computational chemistry domain needs on top of the PaRSEC runtime. This includes an efficient matrix product operation for hybrid architectures (such as Summit) with an irregularly blocked, block-sparse representation of matrices. Moreover, the requirements on this implementation were extremely strict, the matrices are rectangular and extremely large in at least one of their dimensions, such that none of the input matrices could fit in the aggregated memory of all GPUs. The algorithm and a deeper analysis of the results are described in [45].

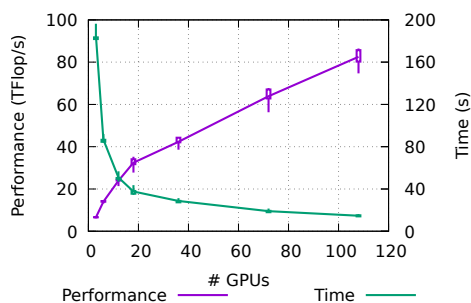


Figure 22: Time to solution and Performance as a function of the number of V100 GPUs on Summit, for the molecule $C_{65}H_{132}$

The strong scaling evaluation shows a parallel efficiency of 35%, with a time to solution at 180s on 3 GPUs, going down to 13s at 118 GPUs. Compared to the state of the art, DBCSR [46] can only run problems that fit in the GPU memory, preventing us to run the same experiment, but experiments on synthetic problems that fit in memory show an improvement by a factor two, while TiledArray [47] cannot leverage the GPUs of Summit without using PaRSEC and would run, on the CPUs of Summit ten times slower.

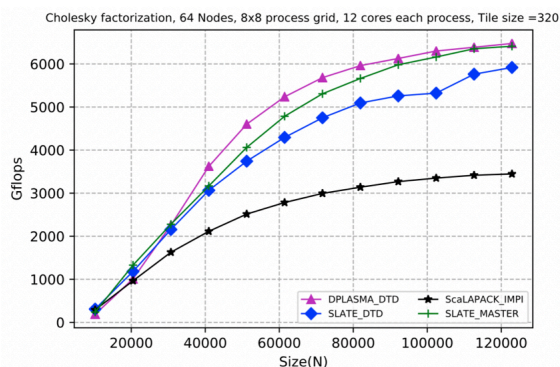


Figure 23: Comparison of DPLASMA and SLATE Cholesky factorization over PaRSEC with SLATE and ScaLAPACK on 64 nodes 12 cores each

in a distributed environment and showed the capability of the SLATE library using a modern fully capable runtime system. This work involved enhancing the insert task interface available in the PaRSEC runtime to map onto the logic of a SLATE algorithm.

Figure 23 compares different implementation of the Cholesky factorization. On one side we have two reference implementations for distributed linear algebra, ScaLAPACK and the current version of the SLATE

Solution Strategy Figure 22 shows a strong scaling performance evaluation of this algorithm, when applied to the main tensor product required by the CCSD method to simulate the electronic structure from first principles. The simulated molecule was $C_{65}H_{132}$, which is a quasi-1-dimensional system and small atomic orbital basis, where the sparsity of tensors is maximized while the optimal (from the data compression perspective) tile size is small. That molecule is representative of applications to 1-d polymers and quasi-linear molecules (such as some proteins), while the choice of the atomic orbital basis is representative of medium-precision simulations in chemistry and condensed phase. Such use case stresses the system and algorithm as it implies a significant sparsity: the largest matrix, while being of square of rank 2, 464,900, has only a density of 3.1%.

An important aspect of the DTE project is to define and prototype scalable domain specific languages that enable a productive expression of parallelism for end-user communities. PaRSEC presents multiple programming interfaces (Parameterized Task Graphs for maximum parallelism, the popular serial task insertion dataflow model to provide direct access to the runtime). In addition, the DTE team is in close contact with application teams to define parallel abstractions that are suitable for their domain usage. Notably, the PaRSEC team has ongoing collaboration with the SLATE linear algebra package and NWChemEx and GAMESS chemistry package teams.

In this context it is interesting to highlight the first step toward the integration of the PaRSEC framework into the SLATE (2.3.3.09) in the context of the shared milestone (STPM11-23). The first prototype of the application ran

library (using OpenMP for intra-node parallelism and MPI for communications). On the other side we have two DSL expressing the same algorithm but using ParSEC as the underlying runtime, the Dynamic Task Discovery (DTD) an approach similar to OpenMP but working on a distributed setting, and a version of the SLATE library where instead of relying on explicit parallelism (OpenMP) and communications (MPI) it rely on implicit dependencies management via ParSEC.

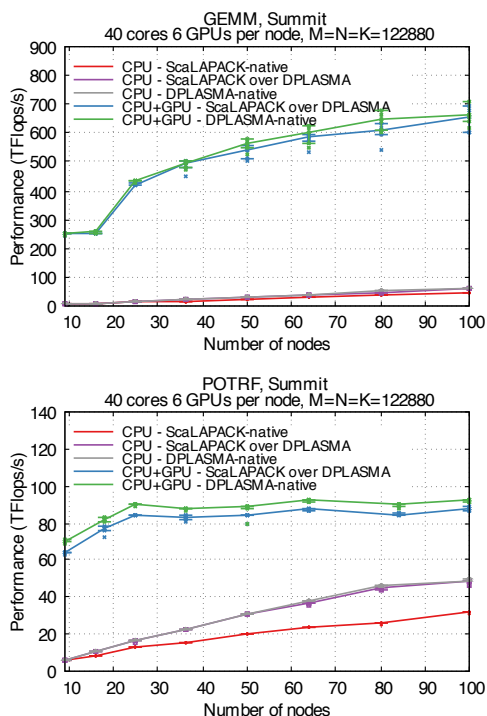


Figure 24: Performance using GPUs of native ScaLAPACK, ScaLAPACK over DPLASMA and native DPLASMA for GEMM and POTRF.

usage of the DPLASMA wrapper introduces a speedup of $1.73\times$ for GEMM (minimum $1.64\times$, maximum $1.88\times$) and $1.49\times$ for POTRF (minimum $1.27\times$, maximum $1.71\times$). When exploiting also the GPUs of the computation nodes, the performance is increased by $16.75\times$ for GEMM (minimum $10.79\times$, maximum $25.68\times$) and by $4.27\times$ for POTRF (minimum $2.77\times$, maximum $6.70\times$). The lower performance of the DPLASMA's POTRF over GPUs is explained because in the current implementation of this algorithm not all the tasks are run on GPUs. However, further improvements of DPLASMA algorithms that will be integrated in the next release of DPLASMA will likely achieved similar performance when used for wrapping the ScaLAPACK routines. Therefore, enabling applications already using the ScaLAPACK API to improve their usage of the available hardware resource with very little effort that is translated in important performance benefits.

Recent Progress The software release (2021.10) provides many new additions to the low-level task runtime, supports for a number of hardware capabilities (AMD GPUs, new device architecture to support Intel GPUs, new communication subsystem to better employ RDMA communications), brings significant improvements to the performance and scalability of the runtime, and addresses many pending issues. ParSEC is used as the default low level task runtime for MADNESS on ECP machines. MADNESS is the task interface and manager used by many of the applications in the NWChemEx ECP project. The ParSEC implementation of the TTG DSL has been greatly improved, reaching performance comparable to other ParSEC DSLs and the state of the art, enabling the wide distribution of this DSL. GPU support has been extended to the DTD DSL, enabling experiments with other ECP applications.

On the software portability and quality side, the ParSEC runtime has been evaluated and updated to

In the context of milestone STPM11-81, the ParSEC team worked to enable the usage DPLASMA as a replacement for ScaLAPACK. This functionality is provided as an independent library which contains a wrapped version of the ScaLAPACK API and hides the ParSEC API from the application while it constructs the structures necessary for the operation with matrices represented on ScaLAPACK memory layout. Users of applications exploiting ScaLAPACK can link against this independent library to run the wrapped routines over DPLASMA-ParSEC, while any other ScaLAPACK function, i.e. that does not have an equivalent provided by DPLASMA, will use the original ScaLAPACK implementation. This approach reduces to a minimum the changes that need to be performed on the ScaLAPACK application, while enabling the exploitation of the algorithms implemented on DPLASMA and the operation over ParSEC for a better exploitation of the available hardware resources.

Figure 24 compare the performance of the ScaLAPACK wrapper extensions against the native DPLASMA and native ScaLAPACK in their typical usage scenarios (one process per core for ScaLAPACK), while DPLASMA tests (native DPLASMA and ScaLAPACK over DPLASMA) use one thread per core and one process per node on the experiments using only the CPU and two processes per node, each exploiting 3 GPUs, are used on the GPU tests. In all cases, the performance achieved by running ScaLAPACK over DPLASMA is more than 90% the performance of native DPLASMA. For the CPU-only experiments, the

compile and run on all Early Access System (Arcticus and Spock), as well as some early platforms based on the new ARM architecture. ParSEC is installable through Spack on all ECP platforms, and available as a system module on Spock. More information about the early access platforms support in Section 4.1.10.

Template Task Graph (TTG) is a new DSL co-developed with members of the NWChemEx team, to address issues in the other DSLs available in ParSEC. TTG mixes the concept original to ParSEC of a Parameterized Task Graph (PTG) to expose a concise and efficient representation of the DAG of tasks, with the dynamicity and data-dependence capabilities of the Dynamic Task Discovery paradigm that is most often used to express task-based systems. A static graph of task classes is built by the program at compile time, but the tasks belonging to this graph are only instantiated at runtime. The originality of TTG lies in the fact that task instantiation in TTG is by nature capable of managing data dependency (i.e. depending on the computation itself a task may or may not become instantiated), and fully distributed. Tasks are instantiated by other tasks, while unrolling the graph of task classes, and they do not require to expose a global knowledge on the DAG, making the approach more scalable than usual DAG discovery.

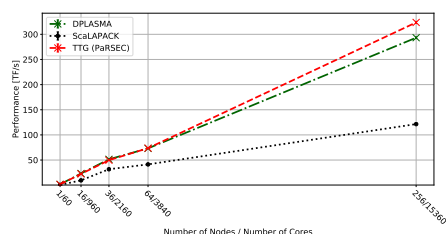


Figure 25: Performance comparison between DPLASMA (PTG DSL), the TTG implementation, and ScaLAPACK, of the Cholesky factorization (double precision), for a fixed amount of memory per node (weak scaling).

In the context of this collaboration, the ParSEC team has provided an efficient implementation of the TTG interface on top of the ParSEC runtime. Figure 25 shows a performance comparison on a 5,632 dual-socket 64-core AMD EPYC 7742 nodes equipped with 256 GB main memory and connected through a Mellanox Infiniband HDR 200 fabric, from 1 to 256 nodes (representing 15,360 cores in total). We compare the TTG implementation of the dense Cholesky factorization with the `pdpotrf` routines available in DPLASMA (thus with the PTG implementation of the same algorithm), and in ScaLAPACK. Each process holds a constant amount of memory, representing a submatrix of $30k \times 30k$ elements. Each node runs 8 processes, and each process is allocated 16 cores. With this weak-scaling experiment, we observe that TTG behaves as efficiently and consistently as PTG. At 15,360 cores,

the TTG implementation even goes slightly faster than the PTG one, due to a better scheduling of the communications that are discovered in a different order. Both task-based and ParSEC-based implementations go significantly faster than ScaLAPACK, which is the expected behavior for a machine with so many cores per node.

Deployment on Early Access Systems The ParSEC runtime as well as its dependent applications can compile and execute on the Spock early access system (EAS) (Figure 26). Recent updates have identified the root cause of prior difficulties with using atomic operations with the default compiler (LLVM-clang-13), which can now be safely used. The GNU compilers had been validated to be correct since last march. The ParSEC runtime can take advantage of the MI100 AMD accelerators (it uses the ROCM software package to orchestrate host-device data transfer), and applications using ParSEC (e.g., DPLASMA dense linear algebra package, Massively Parallel Quantum Chemistry MPQC) have been modified to use rocsolver/rocblas kernels. The performance of some of the kernels provided from rocsolver/rocblas has been found to be underperforming compared to theoretical peak, notably some of the one-sided factorization in rocsolver. Despite these early system limitations, the ParSEC runtime is able to achieve a large proportion of peak performance on AMD accelerated nodes. In addition to single-accelerator support, we validated the multi-GPU scalability of ParSEC on the DGEMM operation (dense matrix-matrix multiply), where we have witnessed that the ParSEC version operates at the same rate as the rocblas DGEMM (as expected), and scales almost perfectly up to 4 GPUs. Similarly, the DPOTRF operation (LLT Cholesky factorization) also shows very good multi-GPU scaling, and ParSEC outperforms by several order of magnitudes the performance of the rocsolver kernel. Performance of the LLT factorization has also been verified to scale to multiple nodes. Looking at a more challenging application pattern, we ran the irregular tensor contraction operation from the MPQC chemistry ECP application. On this application, we showed a very high rate of execution on multiple GPU accelerators, even when the sparsity pattern of the data is high.

The general software environment on Crusher being very similar to Spock, compiling ParSEC on Crusher

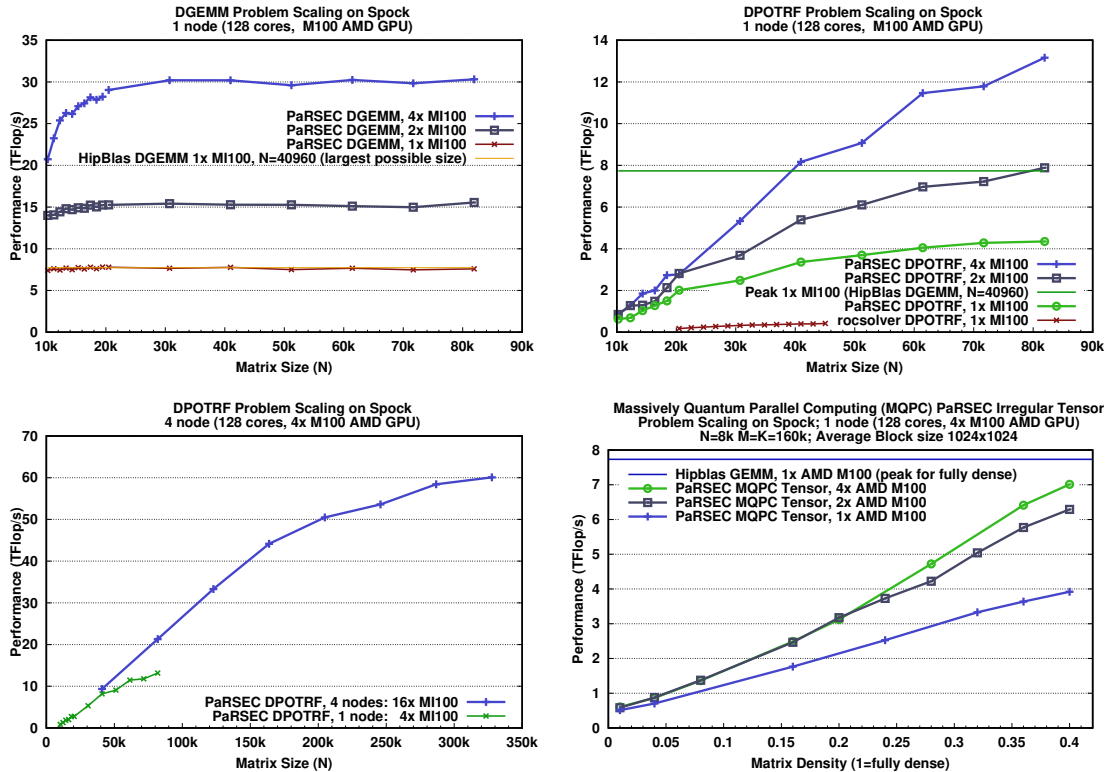


Figure 26: Performance of ParSEC on AMD accelerated system OLCF Spock.

required only minor adaptation of the build scripts (so as to find and load the correct software dependencies from system-installed software). ParSEC has been verified to run on Crusher with basic correctness tests on up to 4 nodes, with multiple GPUs.

On both Crusher and Spock, we identified a limitation of the libSCI BLAS library: it would not allow calling into BLAS operations from all cores when running one thread per-core. This limitation has been communicated to the OLCF support team, but has not been resolved thus far. Our mitigation strategy has been to compile our own copy of OpenBLAS as a substitute that can support enough threads to access the BLAS library.

We have also validated that ParSEC deploys on Nvidia based DOE pre-exascale systems Perlmutter and Summit. All features of ParSEC are supported on these machines, and performance has been validated to be consistent with expectations on a wide range of applications.

Deployment on Intel Xe systems has been tested on the Yarrow system. We also used Yarrow to develop an Xe specific device that manages host-device memory movement. Deployment on the Arcticus EAS is ongoing. We verified that the software can compile and are running basic tests with CPU workloads to verify correctness.

Next Steps To provide programmers with more control over how accelerators are integrated and used by the runtime, a need to provide finer control of the resource usage by the runtime system has arisen. We are developing new APIs to allow the programmers to advise the runtime system with respect to data placement, prefetching, and management of cache. Programming interoperability should not be limited to node-level programming models but should extend to distributed programming. Execution modes where part of the application is expressed in native MPI (including communicating tasks) and other parts using ParSEC DSLs, running above the task system in a tightly coupled manner, are being developed.

The set of tools that come with the ParSEC runtime environment to assess performance, find bottlenecks, improve scheduling and debug the task-based application are being improved to expose the information in a format compatible with TAU, Score-P and other tools that are already familiar to ECP users.

4.1.11 WBS 2.3.1.14 GASNet-EX

Overview GASNet-EX [48] is a portable high-performance communication layer supporting multiple implementations of the Partitioned Global Address Space (PGAS) model. GASNet-EX clients include Pagoda’s PGAS programming interface UPC++ [49, 50] and the Legion Programming System [51, 52] (WBS 2.3.1.08).

GASNet-EX’s low-overhead communication mechanisms are designed to maximize injection rate and network utilization, tolerate latency through overlap, streamline unpredictable communication events, minimize synchronization, leverage hardware support for communication involving accelerator memory, and efficiently support small- to medium-sized messages arising in ECP applications. GASNet-EX enables the ECP software stack to exploit the best-available communication mechanisms, including novel features still under development by vendors. The GASNet-EX communications library and the PGAS models built upon it offer a complementary, yet interoperable, approach to “MPI + X”, enabling developers to focus their effort on optimizing performance-critical communication.

We are co-designing GASNet-EX with the UPC++ development team, along with additional input from the Legion and (non-ECP) Cray Chapel [53, 54] projects.

Key Challenges Exascale systems will deliver exponential growth in on-chip parallelism and reduced memory capacity per core, increasing the importance of strong scaling and finer-grained communication events. Success at exascale demands minimizing software overheads, especially avoiding long, branchy serial code paths; this motivates a requirement for efficient fine-grained communication. The pervasive use of accelerators introduces heterogeneous systems including memory with properties that differ from host DRAM but can still benefit from network offload support during communication. These challenges are exacerbated by application trends; many of the ECP applications require adaptive meshes, sparse matrices, or dynamic load balancing. All of these characteristics favor the use of low-overhead communication mechanisms that can maximize injection rate and network utilization, tolerate latency through overlap, accommodate unpredictable communication events, minimize synchronization, leverage hardware support for communication involving accelerator memory, and efficiently support small- to medium-sized messages. The ECP software stack needs to expose the best-available communication mechanisms, including novel features being developed by the vendor community.

Solution Strategy The PGAS model is a powerful means of addressing these challenges and is critical in building other ECP programming systems, libraries, and applications. We use the term PGAS for models that support one-sided communication, including contiguous and non-contiguous remote memory access (RMA) operations such as put/get and atomic updates. Some of these models also include support for remote function invocation. GASNet-EX [55] is a communications library that provides the foundation for implementing PGAS models, and is the successor to the widely-deployed GASNet library. We are building on over 20 years of experience with the GASNet [48, 56] communication layer to provide production-quality implementations that include improvements motivated by technology trends and application experience.

The goal of the GASNet-EX team is to provide a portable, high-performance PGAS communication layer for exascale and pre-exascale systems, addressing the challenges identified above. GASNet-EX provides interfaces that efficiently match the RDMA capabilities of modern inter-node network hardware and intra-node communication between distinct address spaces. New interfaces for atomics and collectives have enabled offload to current and future network hardware with corresponding capabilities. New interfaces for non-host memory are designed to enable efficient transfers to and from device memories, by leveraging such vendor technologies as GPUDirect RDMA (GDR). Together these design choices and their implementations supply the low-overhead communication mechanisms required to address the requirements of exascale applications.

Figure 27 shows representative results from a paper [55] comparing the RMA performance of GASNet-EX against MPI on multiple systems including NERSC’s Cori and OLCF’s Summit.⁵ These results demonstrate the ability of a PGAS-centric runtime to deliver performance as good as MPI, and often better. The paper presents experimental methodology and system descriptions, which are also available online [48], along with results for additional systems.

⁵The paper’s results from Summitdev have been replaced by more recent (June 2019) results from OLCF’s newer Summit system.

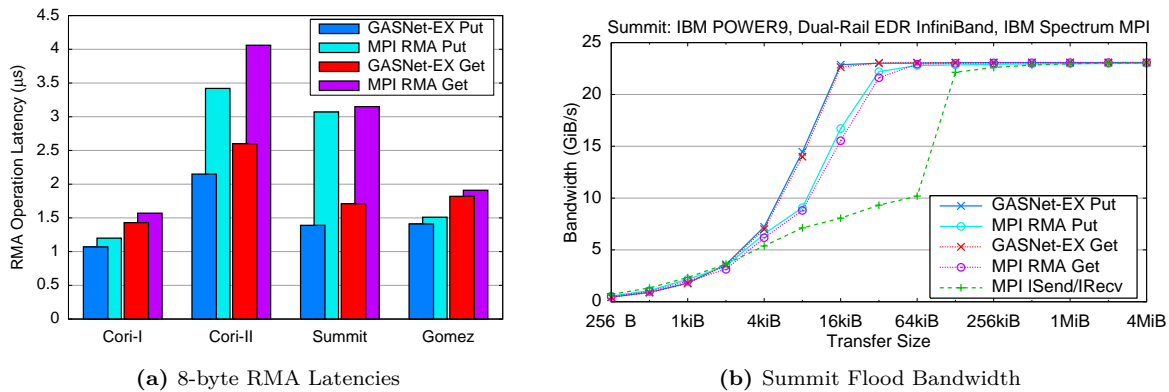


Figure 27: Selected GASNet-EX vs. MPI RMA Performance Results

Figure 27a shows the latency of 8-byte RMA Put and Get operations on four systems, including two distinct network types and three distinct MPI implementations. GASNet-EX’s latency is 6% to 55% better than MPI’s on Put and 5% to 45% better on Get. Algorithms sensitive to small-transfer latency may become practical in PGAS programming models due to these improvements relative to MPI. Figure 27b shows flood bandwidth of RMA Put and Get measured on OLCF’s Summit. GASNet-EX’s bandwidth is seen to rise to saturation at smaller transfer sizes than IBM Spectrum MPI, with the most pronounced differences appearing between 4KiB and 32KiB. Comparison to the bandwidth of MPI message-passing (dashed green) illustrates the benefits of one-sided communication, a major feature of PGAS models.

Recent Progress The most notable work on GASNet-EX in the past year has been in device memory support and tuning.

Device (GPU) Memory Support Memory kinds is the GASNet-EX term for support of communication involving memory other than host memory. In the context of ECP, the primary focus is accelerator devices such GPUs. However, the design is extensible to other accelerators, such as FPGAs, and to storage. Since late 2020, GASNet-EX has leveraged the GDR capabilities of modern NVIDIA GPUs and Mellanox network adapters (such as those on Summit) to perform one-sided RMA involving GPU memory without the overheads of staging through intermediate buffers in host memory. The GASNet-EX APIs for memory kinds have been co-designed with the developers of UPC++ and the Realm runtime layer of the Legion Programming System (WBS 2.3.1.08), and both projects have leveraged this support since their respective late 2020 releases. Performance of GASNet-EX memory kinds, including via Legion and UPC++ benchmarks, is featured on an SC21 research poster [57], and additional UPC++ performance results using memory kinds appear in Figure 28 in Section 4.1.12. The most recent work on memory kinds has extended the implementation in two directions: adding support for the UCX network API and for GPUs using the AMD HIP API. The net result is support for four combinations of network API and GPU vendor.

Tuning We have devoted effort in the past year to improving the performance and memory use of the GASNet-EX runtime. Examples of this work include significant improvements to InfiniBand support which (1) reduce memory consumption at large job sizes and (2) improve scaling to large thread counts in multi-threaded clients. Other recent tuning efforts have reduced the overheads for put, get and atomic updates in intra-node shared memory.

Preliminary Results on Early Access Systems Early access to OLCF’s HPE Cray EX systems, Spock and Crusher, have helped our team to pursue support in GASNet-EX for their respective networks (Slingshot-10 and Slingshot-11) and for efficient RMA transfers to and from AMD GPUs.

Experiences on OLCF’s Spock system helped to enable the recent addition of memory kinds support for the UCX network API and for GPUs using the AMD HIP API. In particular, use of the ROCmRDMA technology allows network hardware to perform RMA operations to and from memory on AMD GPUs without staging

through buffers in host memory. Microbenchmarks collected on Spock of RMA put operations from local host memory to remote GPU memory show bandwidth peaking to the same 11.4GiB/s as for transfers in either direction between host memory at both ends. Meanwhile, RMA get operations from remote GPU memory to local host memory reach a peak bandwidth of 9.3GiB/s. Figure 29 in Section 4.1.12 shows preliminary results for a UPC++ benchmark run on Spock, which illustrate how leveraging the new ROCmRDMA support in GASNet-EX translates into significant performance gains over the prior implementation approach.

Early experiences on OLCF’s Crusher system are driving development of support for the HPE Slingshot-11 network. This work is anticipated to appear in the Spring 2022 release of GASNet-EX. Preliminary results have demonstrated the ability of GASNet-EX to effectively utilize all four network interfaces in a Crusher node, and to deliver peak RMA bandwidths competitive with those HPE has reported for their MPI on comparable transfers.

Next Steps Next steps for each effort have already been identified.

Device (GPU) Memory Support We will continue work in the area of GASNet-EX memory kinds, including the hardening and tuning of the current implementation. As access to relevant systems is secured, we plan to extend support to additional accelerators and networks, with Intel GPUs and the HPE Slingshot network being the most important examples for ECP.

Client-Driven Tuning In collaboration with authors of client runtimes using GASNet-EX (most notably UPC++ and Legion) and their users (such as ExaBiome), we will continue to identify and address any significant scalability issues or performance anomalies which are discovered.

4.1.12 WBS 2.3.1.14 UPC++

Overview UPC++ [50] is a C++ library supporting Partitioned Global Address Space (PGAS) programming [49, 58, 59]. The current UPC++ v1.0 (distinct from an earlier prototype designated V0.1 [60]) is distinguished by three guiding principles. First, all communication is asynchronous, allowing overlap of computation and communication, and encouraging programmers to avoid global synchronization. Second, all communication is syntactically explicit, encouraging programmers to consider the costs of communication. Third, UPC++ encourages the use of scalable data-structures, avoiding non-scalable library features. These principles provide a programming model that can scale efficiently to potentially millions of processors. UPC++ is well-suited for implementing elaborate distributed data structures where communication is irregular or fine-grained. The UPC++ communication interfaces for Remote Memory Access (RMA) and Remote Procedure Calls (RPC) are composable and fit naturally within the context of modern C++.

UPC++ is needed for ECP because it delivers low-overhead communication, efficiently embracing PGAS support offered by modern systems, such as Remote Direct Memory Access (RDMA) offload capabilities of modern network hardware and native on-chip communication between distinct address spaces. UPC++’s low-overhead communication mechanisms can maximize injection rate and network utilization, tolerate latency through overlap, streamline unpredictable communication events, minimize synchronization, leverage hardware support for communication involving accelerator memory, and efficiently support small- to medium-sized messages arising in many applications. The importance of these properties is reinforced by application trends; many ECP applications require the use of irregular data structures such as adaptive meshes, sparse matrices, particles, or similar techniques, and also perform dynamic load balancing. Because of such trends, the UPC++ library provides an essential ingredient for the ECP software stack; enabling it to exploit the best-available communication mechanisms, including novel features being developed by vendors. UPC++ provides seamless and efficient multithreading support, offering a complementary and interoperable approach to “MPI + X”, enabling developers to focus their effort on optimizing performance-critical communication.

Key Challenges As a result of technological trends, the cost of data motion is steadily increasing relative to that of computation. To reduce communication costs, we need to reduce the software overheads and hide communication latency behind available computation. UPC++ addresses both strategies. UPC++ avoids the software overheads associated with MPI, instead relying on the GASNet-EX [48, 55] communication library which is specifically designed and tuned for native PGAS communication using the best-available

hardware mechanisms on each network (see Section 4.1.11 on GASNet-EX, which is being co-designed). UPC++ supports asynchronous communication via one-sided RMA and RPC.

Solution Strategy The UPC++ project has two primary thrusts.

Increased Performance through Reduced Communication Costs UPC++ leverages the underlying GASNet-EX communication library to deliver efficient, low-overhead RMA and RPC on HPC systems. This includes accelerated transfers to and from GPU memory. UPC++ features encourage aggressively asynchronous execution, enabling applications to overlap the latency of communication with available computation or additional communication.

Improved Productivity UPC++’s treatment of asynchronous execution utilizes futures and promises, which simplify the management of asynchrony. The `upcxx::copy` API provides concise expression of RMA involving any combination of shared host and GPU memory, without the need for the application to stage GPU data through host memory buffers.

The PGAS model and one-sided RMA communication offered by UPC++ benefit application performance by mapping tightly onto the RDMA mechanisms supported by the network hardware. GASNet-EX provides the thin middleware needed to implement RMA efficiently on a variety of platforms, from laptops to supercomputers. One-sided communication provides an additional benefit of decoupling synchronization from data motion, avoiding the fine-grained synchronization overheads of message passing.

UPC++’s Remote Procedure Call (RPC) enables the programmer to easily execute code on remote processes. RPC is useful in managing access to complicated irregular data structures, and in expressing asynchronous task execution, where communication patterns are data-dependent and hence difficult to predict.

As one example of how our approach is applicable to real application kernels, we modified a GPU-enabled 3D heat conduction example from a Kokkos tutorial to convert the inter-node communication from MPI message passing to UPC++ RMA [61]. The primary goal of this study was to demonstrate that comparable programming effort with UPC++ and MPI could yield comparable performance. However the finding was that with both versions written naively, the UPC++ RMA version gave measurably superior performance. Figure 28 shows a strong scaling study on OLCF’s Summit, comparing performance using CUDA-aware IBM Spectrum MPI versus UPC++ with the CUDA memory kind.

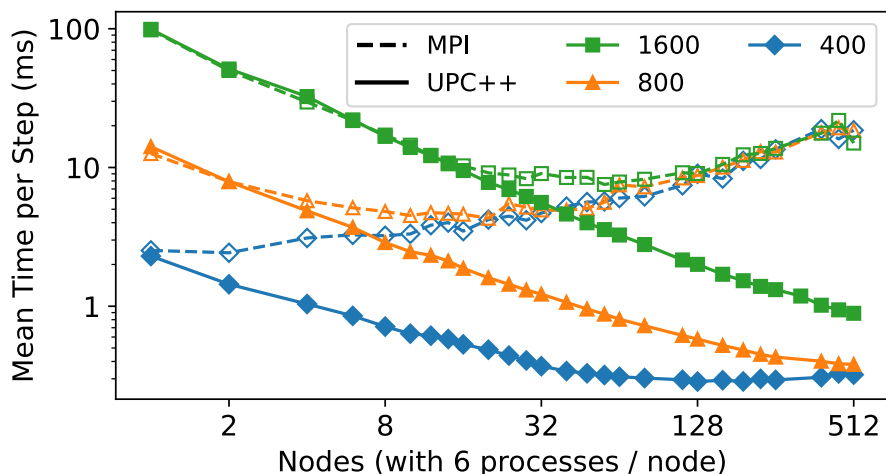


Figure 28: Mean run time per step (lower is better) in a strong scaling study comparing UPC++ (solid series) and MPI versions (dashed series) of a GPU-enabled 3D heat conduction example code, run on OLCF’s Summit for three problem sizes (given as length of one edge of the cubic domain).

Recent Progress The most notable work in the past year has been in three areas:

Memory Kinds UPC++ memory kinds provide a unified abstraction for communication between combinations of device (e.g. GPU) and host memory, possibly remote. By unifying the expression of data transfer among the various memories in a heterogeneous system, these abstractions enable ECP applications to utilize accelerators within UPC++’s PGAS model. The abstraction enables hardware offload (when available) of device data transfers, eliminating the need to stage transfers through intermediate buffers in host memory. The global pointer representation transparently tracks device information, eliminating the need for expensive address space queries in critical paths. Support for such offload on the OLCF’s Summit was initially demonstrated in October 2020, and subsequent releases have hardened and tuned the implementation.

Training and Outreach The UPC++ team continues outreach efforts, including a half-day UPC++ tutorial at SC21 [62] (for the second consecutive year).

Productivity and Performance Having completed the core specification and implementation of UPC++, we have shifted focus toward addressing improvements to productivity and performance, especially in response to stakeholder feedback. The most significant examples are deployment of eager notification [63] and instruction-level tuning along common critical paths.

Preliminary Results on Early Access Systems We are using OLCF’s Spock (HPE Cray EX) system to implement support in UPC++ for efficient RMA operations targeting the memory in GPUs using the AMD HIP API. This work leverages recently completed work in GASNet-EX which uses the ROCmRDMA technology to allow network hardware to perform RMA operations to and from memory on AMD GPUs without staging through buffers in host memory. This work is anticipated to appear in the Spring 2022 release of UPC++.

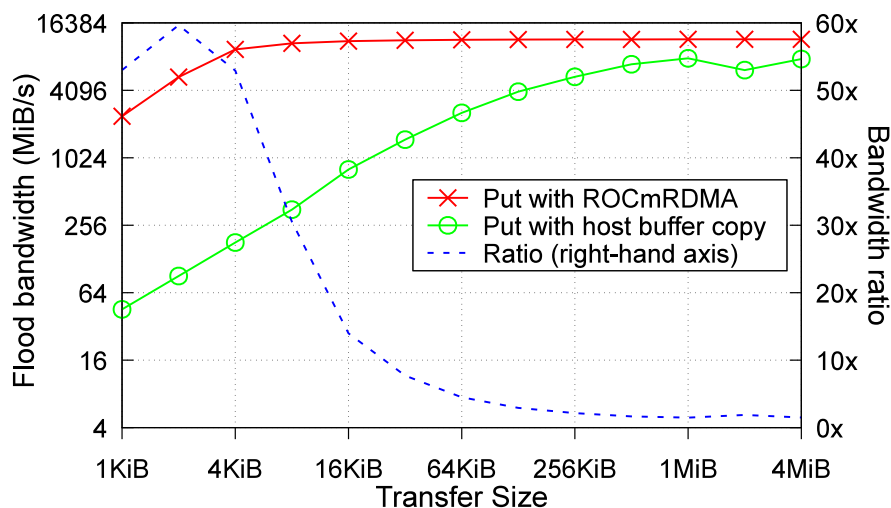


Figure 29: Preliminary flood bandwidth on OLCF’s Spock of RMA put operations from local host memory to remote GPU memory, implemented with and without use of the ROCmRDMA capability in GASNet-EX, along with the bandwidth ratio.

Figure 29 shows preliminary results of this work. Solid lines show the flood bandwidth of put operations from local host memory to remote GPU memory, collected using the work-in-progress to support AMD GPUs in UPC++. The X symbols mark data with ROCmRDMA support enabled in GASNet-EX, while O symbols mark data with that support disabled. In the latter case, UPC++ uses the HIP API to stage GPU data through host memory at the remote host. The dashed line shows the ratio on a linear scale, highlighting the magnitude of the improvement which is not immediately evident on the logarithmic bandwidth scale. For transfer sizes up to 16 KiB the improvement is over 10x. For smaller transfers the improvement can be nearly 60x, and for larger transfers the improvement is consistently over 1.5x. Benchmark results for a get operation which fetches from remote GPU memory are qualitatively similar.

Next Steps Next steps for each effort have already been identified.

Memory Kinds We will continue development of memory kinds. The past releases of UPC++ targeted the hardware in Summit, while the Spring 2022 release is anticipated to extend support to the hardware in OLCF’s Spock and Crusher systems. Additional future work will include other accelerators scheduled to appear in DOE’s exascale systems. For all accelerators, transfers will be offloaded to available hardware capabilities by leveraging the parallel efforts in GASNet-EX.

Productivity and Performance With the help of our stakeholders, we will continue to identify and address portions of UPC++ where performance tuning is most needed and/or beneficial. Similarly, we will continue to work with stakeholders to identify and implement features which improve productivity.

Outreach We will continue to engage with our users, such as by circulation of working group drafts of new features to solicit feedback.

4.1.13 WBS 2.3.1.16 SICM

Overview The goal of this project is to create a universal interface for discovering, managing and sharing within complex memory hierarchies. The result will be a memory API and a software library which implements it. These will allow operating system, runtime and application developers and vendors to access emerging memory technologies. The impact of the project will be immediate and potentially wide reaching, as developers in all areas are struggling to add support for the new memory technologies, each of which offers their own programming interface.

Key Challenges The challenge SICM addresses is how to program the deluge of existing and emerging complex memory technologies on HPC systems. This includes the High Bandwidth Memory (HBM), MCDRAM (on Intel Knights Landing), NV-DIMM, PCI-E NVM, SATA NVM, PCM, memristor, and 3Dxpoint. Also, near node technologies, such as disaggregated memory or network attached memories, have been proposed in exascale memory designs. Current practice depends on ad hoc solutions rather than a uniform API that provides the needed specificity and portability. This approach is already insufficient and future memory technologies will only exacerbate the problem by adding additional proprietary APIs.

Solution Strategy The SICM solution is to provide a unified two-tier node-level complex memory API. The target for the low-level interface are system and runtime developers, as well as expert application developers that prefer full control of what memory types the application is using. The SICM high-level interface employs application profiling and analysis to direct data management across complex memory hierarchy. The earlier approach is based on an offline profiling approach called MemBrain. The team extended the SICM high-level interface with facilities to profile, analyze, and migrate application data within the same program run. This online approach, Figure 30, reduces user burden and achieves similar performance to the earlier offline approach after a short initial startup period. The low-level interface is primarily an engineering and implementation project. The solution it provides is urgently needed by the HPC community; as developers work independently to support these novel memory technologies, time and effort is wasted on redundant solutions and overlapping implementations. Adoption of the software is focused on absorption into existing open source projects such as hwloc, KOKKOS, Umpire, CLANG/LLVM, OpenMP, and Jemalloc. Additionally, SICM is developing Metall, a persistent memory allocator designed to provide developers with an API to allocate custom C++ data structures in both block-storage and byte- addressable persistent memories (e.g., NVMe and Intel Optane DC Persistent Memory) beyond a single process lifetime. Metall relies on a file-backed mmap mechanism to map a file in a filesystem into the virtual memory of an application, allowing the application to access the mapped region as if it were regular memory which can be larger than the physical main-memory of the system.

Recent Progress Recent progress includes developments in the low-level interface, the high-level API, and the high-level graph interface.

Low-Level Interface Finished refactor of low-level interface supporting memory arenas on different memory types. Added support for Umpire, OpenMP and KOKKOS. Investigating features need to fully support these runtimes, currently KOKKOS and the Nalu-wind ECP application. SICM now supports Intel Optane memory, the first NVM memory that can be used as an extension of traditional DRAM memory. Pull requests have been developed for OpenMP/CLANG/LLVM, KOKKOS and Umpire. the patches to Clang/LLVM/OpenMP turn OpenMP memory spaces in OpenMP 5.x into SICM library calls in the LLVM/OpenMP runtime. The same codepath that supports memkind library was refactored to support multiple custom memory allocators – more general than just SICM support. SICM currently supports `pragma openmp allocate` with memory types: `omp_ (default, large_cap, const, high_bw, and low_lat) _mem_spaces` and supports KNL, Optane, testing on Sierra/Summit. SICM continues to target the early access systems for Frontier and Aurora.

High-Level API SICM has employed application profiling and analysis to direct data management across complex memory hierarchy, the team extended the SICM high-level interface with application-directed data tiering based on the MemBrain approach which is more effective than an unguided first touch policy. The impact of using different data features to steer hot program data into capacity-constrained device tiers was modeled.

High-Level Graph Interface For the Metall high-level interface focusing on graph applications, work has used miniVite, and ECP graph proxy application. miniVite has been modified to store and reuse graph data using Metall. Graph generation has been a bottleneck in this application. Metall can use mmap and UMap (user-level mmap library in Argo PowerSteering project) underneath to enhance its performance and capability.

Next Steps Next steps for each effort have already been identified.

Low-Level Interface Focus on performance of support for runtimes and adding feature requested to support Umpire, OpenMP KOKKOS, and MPI and address the slow move pages implementation in the Linux kernel. Initial efforts to accelerate move pages have not been successful and progress has been hampered by current work environment. The Linux kernel modifications for page migration are in collaboration with ECP project Argo (2.3.1.19) and RIKEN research center in Japan. Collaborate with applications to enable use of heterogeneous memory on ECP target platforms. Additionally, we plan to reconnect with the hwloc team on memory topology discovery.

High-Level API For the high-level interface analysis work, the team will adapt their approach and toolset to remove dependence on source code analysis and recompilation of the target application. The goal is to enable guided data management for arbitrary processes, even if source code is not available. The new tools will also integrate with the Linux memory manager in order to direct data management for multiple processes running on the same platform simultaneously.

High-Level Graph Interface Metall will continue its collaboration with the ExaGraph team to integrate Metall into an ExaGraph application or benchmark to enable persistent and external-memory data structure support.

4.1.14 WBS 2.3.1.17 Open MPI for Exascale (OMPI-X)

Overview The OMPI-X project ensures that the Message Passing Interface (MPI) standard, and its specific implementation in Open MPI meet the needs of the ECP community in terms of performance, scalability, and capabilities or features. MPI is the predominant interface for inter-process communication in high-end computing. Nearly all of the ECP application (AD) projects (93% [64]) and the majority of ST projects (57% [64]) rely on it. With the impending exascale era, the pace of change and growing diversity of HPC architectures pose new challenges that the MPI standard must address. The OMPI-X project is active in the MPI Forum standards organization, and works within it to raise and resolve key issues facing ECP applications and libraries.

Open MPI is an open source, community-based implementation of the MPI standard that is used by a number of prominent HPC vendors as the basis for their commercial MPI offerings. The OMPI-X team is comprised of active members of the Open MPI community, with an extensive history of contributions to this

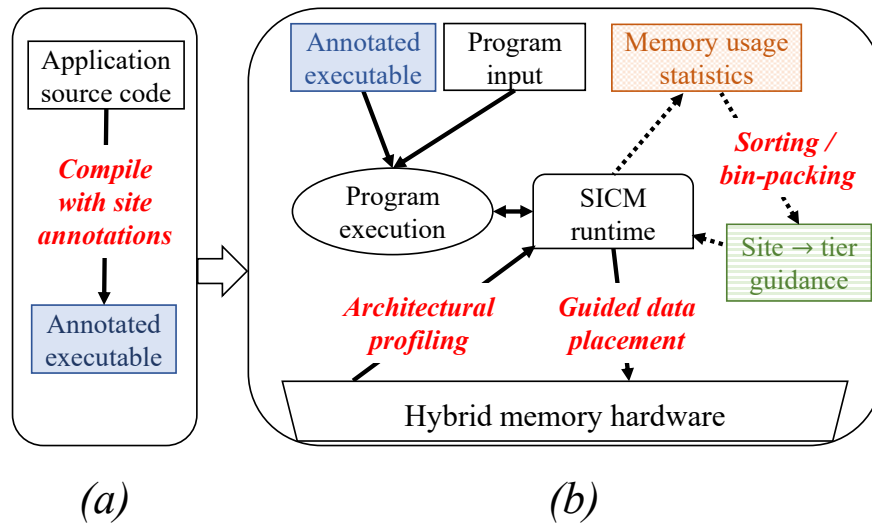


Figure 30: Data tiering with online application guidance. (a) Users first compile the application with a custom pass to insert annotations at each allocation call site, (b) Program execution proceeds inside the SICM runtime layer, which automatically profiles memory usage behavior, converts it into tier recommendations for each allocation site, and enforces these recommendations during program execution. In (b), interactions and operations drawn with dashed lines only occur at regular, timer-based intervals, while the solid lines correspond to activities that can occur throughout the program execution [4].

community. The OMPI-X project focuses on prototyping and demonstrating exascale-relevant proposals under consideration by the MPI Forum, as well as improving the fundamental performance and scalability of Open MPI, particularly for exascale-relevant platforms and job sizes. MPI users will be able to take advantage of these enhancements simply by linking against recent builds of the Open MPI library.

In addition to MPI and Open MPI, the project also includes two other products, which are less visible to the end user, but no less important. PMIx (Process Management Interface for Exascale) provides facilities for scalable application launch, process wire-up, resilience, and coordination between runtimes. It originated as a spin-off from the Open MPI community, but is now developing a community of its own as adoption grows. Starting in FY20, Qthreads (formerly WBS 2.3.1.15) is also part of the OMPI-X project. Qthreads is a user-level lightweight asynchronous thread library particularly focused on improving support for multithreading in the context of network communications. Both PMIx and Qthreads help the OMPI-X project address key issues of performance and capability for exascale applications.

Key Challenges A number of aspects of exascale levels of computing pose serious challenges to the tried and true message passing model presented by MPI and its implementations, including Open MPI. Keeping pace with changes in HPC architecture is a major challenge. Examples include massive node-level concurrency, driving the growth of MPI+X programming approaches, and the complex memory architectures, which make the placement of data within memory more important. In the near-term, with GPUs dominating the exascale environment, how code running on the GPUs interacts with MPI and inter-process communications must also be addressed. This will require both changes to the standard and changes and improvements within implementations. Performance and scalability become both more important and more challenging as node counts increase and memory per MPI rank trends downward. Finally, as we identify solutions to these challenges that must be implemented within the MPI *standard* rather than particular MPI libraries, we must work within the much larger and broader MPI community that may not always be attuned to the needs of computing at the largest scales.

Solution Strategy The OMPI-X project is working across a number of fronts to address these challenges.

Runtime Interoperability for MPI+X and Beyond MPI is increasingly being used concurrently with other runtime environments. This includes both MPI+X approaches, where X is most often a threading model, such as OpenMP, as well as the use of multiple inter-process runtimes within a single application. Concerns include awareness of other runtimes, cooperative resource management capabilities, and ensuring that all concurrently active runtimes make progress. We are developing APIs and demonstrating capabilities for interoperability in both MPI+X and multiple inter-process runtime situations.

Extending the MPI Standard to Better Support Exascale Architectures The MPI community is standardizing a number of ideas that are particularly important to supporting the architectural and system size characteristics anticipated for exascale. Partitioned communications (previously called *Finepoints*) deal with the growing use of threading for node-level concurrency, in combination with MPI. Sessions increases the flexibility of MPI semantics in a number of areas, which in turn can open opportunities for enhanced scalability, as well as easier support for multicomponent applications such as coupled multiphysics simulations. Error management and recovery capabilities are key to ensuring that applications can detect and respond effectively when errors, inevitably, occur during execution. We are helping to drive incorporation of these and other ideas into the MPI standard by developing prototypes and working with ECP teams and the broader community to demonstrate their feasibility and value.

Open MPI Scalability and Performance As we push the scale of both hardware and applications, we stress MPI implementations and expose areas that need to be improved. OMPI-X is targeting key components within Open MPI, such as threading capabilities, memory usage, remote memory access (RMA), tag matching, and other areas, for improvements in both scalability and performance.

Supporting More Dynamic Execution Environments We are developing and implementing strategies to help MPI applications better deal with topological process layout preferences and contention in the network.

Resilience in MPI and Open MPI Concerns about system and application resilience increase as either scales in size. Our goal in this area is to ensure that MPI, Open MPI, and PMIx provide not only support for simplified recovery for the widely used checkpoint/restart fault tolerance strategy, but also the building blocks to support more general error management and recovery by applications (the evolution of the User-Level Fault Mitigation concept). We work within the MPI Forum, implement, and train users on resilience strategies.

MPI Tools Interfaces Several interfaces within the MPI standard are primarily used to support performance and correctness tools. The MPI Forum is in the process of making significant revisions and extensions to these interfaces. We will track the discussions in the Forum and provide prototype implementations within Open MPI to facilitate evaluation and provide feedback. We will work with the ECP community, including tool developers, to make additional data available through the MPI_T interface.

Quality Assurance for Open MPI We are enhancing the Open MPI testing infrastructure, adding tests to reflect ECP requirements, and instantiating routine testing on systems of importance to ECP, both for correctness and performance.

Recent Progress The MPI Forum finalized and released version 4.0 of the standard in June 2021. The OMPI-X team helped drive the incorporation a number of new features and capabilities considered important for exascale applications. These include sessions and a number of error management and recovery features based on the long-standing User-Level Fault Mitigation (ULFM) concept. Partitioned communications (Figure 31) is another major addition to the standard which is designed to benefit massively-threaded environments, such as GPU accelerators, by supporting incremental completion of the communication as portions of the buffer become ready. These capabilities have at least prototype-level implementations available in the Open MPI library, allowing interested project teams to start exploring the new capabilities.

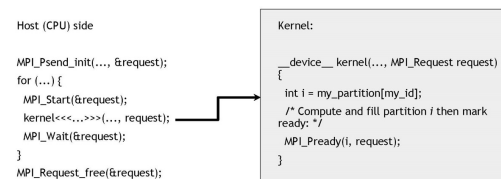


Figure 31: Schematic code illustrating use of partitioned communications from a GPU kernel. Host code (left) sets up communication, GPU kernel (right) triggers send (MPI_Pready).

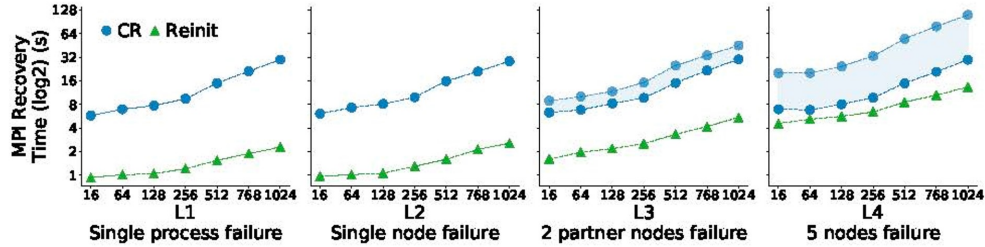


Figure 32: Comparison of recovery times for traditional checkpoint/restart (blue) and Reinit (green) for the HPCG benchmark. Each graph represents different types/severities of failures.

As the MPI Forum continues its work beyond 4.0, the OMPI-X team continues its strategic involvement on topics that did not make the 4.0 release of the standard or further enhancements to 4.0 features to improve support for high-end systems and the ECP community, particularly in the context of resilience. The ULFM proposal has been updated to reflect those features that have been incorporated into 4.0, as well as the discussions within the Forum. For the complementary Reinit simplified checkpoint/restart proposal (Figure 32), we have carried out a formal verification on the recovery algorithm. This ensures that the protocol, which we plan to propose for a future version of MPI, correctly handles the recovery phase of a failure response, including correct propagation of notifications, absence of deadlocks, and proper termination. During the past year, the Reinit recovery protocol has been formally verified and a prototype implementation has been developed in Open MPI. Performance evaluation of the approach show it offering superior performance to traditional checkpoint/restart for the HPCG benchmark against different types of failures (Fig. 32).

As part of the larger Open MPI community, we are working towards a major-version release (v5.0.0), planned for late CY2021. This release will include many contributions of the OMPI-X team, including the beginnings of our work towards full support for MPI 4.0, user-level fault mitigation and FP16 support (both of which we will be advocating for future MPI standards), and performance enhancements such as SIMD support for MPI reductions, and improvements to collective and atomic operations.

To support highly-threaded MPI+X use cases primarily on the CPU side, the OMPI-X project has teamed with the MPICH and Argobots ECP teams (WBS 2.3.1.07 and 2.3.2.11) to define interfaces in both MPICH and Open MPI to support the use of lightweight user-level threading (ULT) libraries such as Qthreads or BOLT as an alternative to Pthreads. The new ULT support, now integrated into both libraries, offer significantly improved performance compared to the traditional Pthreads implementation (Figure 33).

Another key product that the OMPI-X project supports is the Process Management Interface for Exascale (PMIx). PMIx includes both a community standard and a reference implementation, OpenPMIx. Open MPI relies on the PMIx standard and the PRRTE reference runtime for process startup, wireup, and runtime coordination. It is also used and implemented by a variety of other libraries and projects, including LLNL’s Flux resource manager (WBS 2.3.6.02), which is in turn part of the ExaWorks toolkit (WBS 2.3.5.10). During the past year, version 4.0 of the PMIx standard was released, as well as version 4.1.0 of OpenPMIx and version 2.0.0 of PRRTE.

In addition to these activities, we continue to support quality assurance of the Open MPI code base through more and better testing. The Open MPI testing infrastructure, MTT, continues to be improved for flexibility and capability. One of this year’s noteworthy efforts was the addition of the bueno application test harness to the testing system. The facilitates incorporating third-party test cases (i.e. based on user

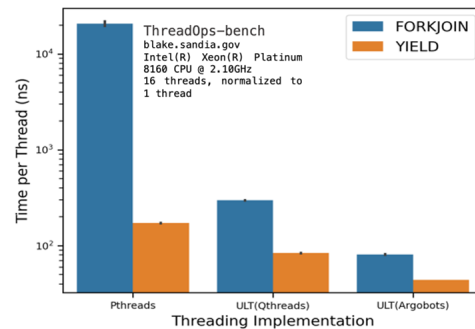


Figure 33: Illustration of relative costs of PThreads vs ULT threads (Qthreads, Argobots) using ThreadOps benchmark for fork-join and yield operations.

applications) into the routine testing process without having to commit them to the Open MPI testing repository. We are also working to establish more frequent testing regimes on key DOE hardware platforms.

Next Steps The delivery of the first HPE Slingshot-11 (SS-11) network, as part of the Frontier system, at the beginning of FY22 finally affords us the opportunity to start our work to ensure that Open MPI provides good support for SS-11, which we expect to be the focus of significant effort this year. SS-11 will be used on both Frontier and Aurora, so much of the work will support both systems, though there will be differences in on-node (particularly GPU-to-GPU) transfers due to the different GPU accelerators. The different workload management solutions on the two platforms will also need to be supported. Additionally, we will be in a much better position to work with application teams who can benefit from the new capabilities embodied in MPI 4.0, but who have been reluctant to adopt them until they were officially part of the standard. We will likewise continue to identify and improve performance and scalability bottlenecks and drive forward on the core themes outlined in our Solution Strategy.

Early Access System Experience Most of the early access systems have utilized InfiniBand interconnects, which are well-supported by Open MPI (the HPE Slingshot-10 interconnect is also InfiniBand). The Frontier test and development system Crusher was the first system available with the new Slingshot-11 network. A subset of the OMPI-X team obtained access to Crusher in December, and have been working to port Open MPI to the SS-11 interconnect. The basic approaches available leverage either Open Fabric providers or the UCX communication infrastructure, or some combination. So far, we have confirmed basic point-to-point functionality using the HPE-provided CXI Open Fabric provider, and partial intra-node shared memory functionality through the UCX communications framework. We plan to continue exploring all three paths until we arrive at a solution that provides full functionality with the expected performance.

The Qthreads user-level lightweight threading package is not dependent on the network, nor on the GPUs so it is easier to port. It has been confirmed to work on Spock (the Frontier early access system). Challenges getting the Qthreads team access to ANL early access systems have delayed demonstration of the functionality there, but we do not anticipate any problems. The team just recently obtained access to Arcticus and plans to test Qthreads there shortly.

4.1.15 WBS 2.3.1.18 RAJA/Kokkos

Introduction The RAJA/Kokkos sub-project is a new combined effort intended to focus on collaborative development of backend capabilities for the Aurora and Frontier platforms. The formation of this project is significant in that it brings two independent teams, RAJA (primarily from LLNL) and Kokkos (primarily from Sandia), to work on a common goal. This project also enhances interactions with staff from other labs, in particular Argonne and ORNL, to help integrate RAJA and Kokkos into the software stack and applications at the respective leadership computing facilities. The remainder of this section is focused on the Kokkos-specific activities. A description of RAJA is provided in the NNSA/LLNL section 2.3.6.02.

Overview The Kokkos C++ Performance Portability Ecosystem is a production-level solution for writing modern C++ applications in a hardware-agnostic way. Started by Sandia, it is now supported by developers at the Argonne, Berkeley, Oak Ridge, Los Alamos, and Sandia National Laboratories as well as the Swiss National Supercomputing (Centre). It is now used by more than a hundred HPC projects, and Kokkos-based codes are running regularly at-scale on half of the top ten supercomputers in the world. The EcoSystem consists of multiple libraries addressing the primary concerns for developing and maintaining applications in a portable way. The three main components are the Kokkos Core Programming Model, the Kokkos Kernels Math Libraries and the Kokkos Tools. Additionally, the Kokkos team is participating in the ISO C++ standard development process, to get successful concepts from the Kokkos EcoSystem incorporated into the standard. Its development is largely funded as part of the ECP, with a mix of NNSA ATDM and SC sources.

Key Challenges One of the biggest challenges for the Exascale supercomputing era is the proliferation of different computer architectures, and their associated mechanisms to program them. Vendors have an incentive to develop their own models in order to have maximum freedom of exposing special hardware capabilities, and potentially achieve vendor-lock-in. This poses the problem for applications that they may

need to write different variants of their code for different machines - an effort which can be simply not feasible for many of the larger application and library projects.

The Kokkos project aims at solving this issue by providing a programming solution which provides a common interface build upon the vendor specific software stacks. There are a number of technical challenges associated with that. First an abstraction must be designed which is restricted enough to allow mapping to a wide range of architectures while allowing exploitation of all the hardware capabilities provided by new architectures. Secondly, the development of support for a new architecture may take significant resources. In order to provide a timely solution for applications in line with the availability of the machine, CoDesign collaborations with the vendors are critical. At the same time software robustness, quality and interface stability is of utmost importance. In contrast to libraries such as the BLAS, programming models permeate the entire code base of an application, and are not isolated to simple call sites. API changes thus would require a lot of work inside of the users code base. A fourth challenge is that in order to debug and optimize the code base tools are required to gain insights into the application.

Besides the technical challenges, a comprehensive support and training infrastructure is absolutely critical for a new programming model to be successful. Prospective users must learn how to use the programming model, current users must be able to bring up issues with the development team and access detailed documentation, and the development team of the model must be able to continue technical efforts without being completely saturated with support tasks. The latter point became a significant concern for the Kokkos team with the expected growth of the user base through ECP. Already before the launch of ECP, there were multiple application or library teams starting to use Kokkos for each developer on the core team – a level not sustainable into the future without a more scalable support infrastructure. This issue was compounded by the fact that Kokkos development was funded through NNSA projects, making it hard to justify extensive support for open science applications.

Solution Strategy To address the challenges the Kokkos team is developing a set of libraries and tools which allow application developers to implement, optimize and maintain performance portable codes. At its heart the EcoSystem provides the Kokkos Core Programming Model. Kokkos Core is a programming model for parallel algorithms that use many-core chips and share memory among those cores. The programming model includes abstractions for frequently used parallel execution patterns, policies that provide details for how those patterns are executed, and execution spaces that denote on which execution agents the parallel computation is performed. Kokkos Core also provides fundamental data structures with policies that provide details for how those data structures are laid out in memory, memory spaces that denote in which memory the data reside, and data access traits conveying special data access semantics. The model works by requiring that application development teams implement their algorithms in terms of Kokkos’ patterns, policies, and spaces. Kokkos Core can then map these algorithms onto the target architecture according to architecture-specific rules necessary to achieve best performance.

Kokkos Kernels is a software library of linear algebra and graph algorithms used across many HPC applications to achieve best (not just good) performance on every architecture. The baseline version of this library is written using the Kokkos Core programming model for portability and good performance. The library has architecture-specific optimizations or uses vendor-specific versions of these mathematical algorithms where needed. This reduces the amount of architecture-specific software that an application team potentially needs to develop, thus further reducing their modification cost to achieve “best in class” performance.

Kokkos Tools is an innovative “plug in” software interface and a growing set of performance measurement and debugging tools that plug into that interface for application development teams to analyze the execution and memory performance of their software. Teams use this performance profiling and debugging information to determine how well they have designed and implemented their algorithms and to identify portions of their software that should be improved. Kokkos Tools interfaces leverage the Kokkos Core programming model interface to improve an application developer’s experience dramatically, by forwarding application specific information and their context within the Kokkos Core programming model to the tools.

Kokkos Support addresses the challenges of establishing, growing and maintaining the user community. First and foremost, it provides explicit means for supporting all DOE ECP applications. A main component of that is funding for local Kokkos experts at the Sandia, Oak Ridge, Argonne, Berkeley, and Los Alamos laboratories which can serve as direct contacts for local applications and the users of the leadership computing

facilities. Secondly, the project develops and maintains a reusable support infrastructure, which makes supporting more users scalable and cost effective.

The support infrastructure consists of GitHub wiki pages for the programming guide and API reference, GitHub issues to track feature requests and bug reports, a Slack channel for direct user-to-user and user-to-developer communication, and tutorial presentations and cloud-based Kokkos hands-on exercises.

The Kokkos Team is also actively engaging the ISO C++ Committee, where it provides about a third of the members interested in HPC. This strong engagement enables the team to lead or contribute to numerous proposals. Among those proposals the team leads are abstractions for multi dimensional arrays based on Kokkos View, atomic operations on generic types and linear algebra algorithms based on Kokkos Kernels, which cover not only the classic Fortran BLAS capabilities, but also batched BLAS and mixed precision linear algebra. The team also has a central role in the primary proposal introducing heterogeneous computing into the C++ standard via the executors concept.

Furthermore, certain areas of common needs between RAJA and Kokkos have emerged. To avoid duplicated efforts and to leverage possible synergies, the two teams are working together to develop advanced atomic support with memory order and memory scope exposure common metaprogramming facilities, optional integration of Umpire memory pools into Kokkos, integration of Kokkos Tools callback mechanisms into RAJA, and an extension of the RAJA performance test suite to include Kokkos variants.

Recent Progress The Kokkos project now consists of an integrated developers team spanning five DOE National Laboratories. In particular both NNSA and SC funded developers are working based off the same task and code management system, use a shared slack channel, and attend a common weekly team meeting. This ensures that no duplication of effort happens, and makes Kokkos a true inter-laboratory project.

Kokkos is used by many applications in production across the entire spectrum of DOE's super computers. Support for current production platforms is mature and stable. Work on supporting the upcoming Exascale platforms is almost complete, with new backends for AMD GPUs and Intel GPUs working. Initial application tests were successfully conducted with projects such as EXAALT/LAMMPS, ArborX, Trilinos and Cabana. Performance optimization work with applications is now ongoing, with a particular focus on the ORNL Frontier super computer.

A training course was developed called *The Kokkos Lectures*, which consists of about 15 hours of recorded lessons and over 20 hands-on exercises. It is available at <https://kokkos.link/the-lectures>. The <https://kokkosteam.slack.com> channel has grown dramatically in use, with about 500 users at the end of 2020, of which more than 150 are active in any given week. The team finished developing a full API documentation as well as adding use case descriptions for common patterns found in applications.

Auto-tuning is now available as an integrated capability into Kokkos with user facing hooks, which allow for the development of custom tuning tools.

At the C++ committee, the MDSPAN proposal is now in wording review - meaning that the technical design is approved. MDSPAN will be able to provide all the core capabilities of Kokkos View. This includes compile and runtime extents, customizable layouts, and data access traits. The extension to heterogeneous memory can be achieved by trivial extensions. Furthermore, the `atomic_ref` proposal was voted into C++20. This capability will provide atomic operations on generic allocations as powerful as Kokkos' atomic operations. In particular it allows atomic operations on types independent of their size, and not just the ones native in the hardware. Another effort making steady progress is the proposal for linear algebra functions. It entails functionality covering all of BLAS 1, 2, and 3, but extends it to any scalar types (including mixing of scalar types) and batched operations. The proposal has passed an initial major design review in the committee and is expected to be forwarded for wording review soon. The Kokkos team was also able to gain co-authors from NVIDIA, Intel and AMD - providing significant support from the leading hardware vendors.

Implementations of those proposals are now available on GitHub. The new atomic operations implementation is hosted at <https://github.com/desul/desul> which serves as the common utility repository for both Kokkos and RAJA. RAJA integrated the Kokkos Tools callback interface, which allows it to leverage investment into tools made by the Kokkos effort.

Preliminary Experiences on Early Access systems Both Kokkos and RAJA have done extensive work on early access systems for Frontier, Aurora and El Capitan. On MI100s (e.g. Spock) both Kokkos and

RAJA pass all their tests and numerous applications are fully functional. Testing on MI100 is now part of the regular CI. On MI250s (e.g. Crusher) Kokkos passes all its tests, with RAJA passing more than 99%.

Kokkos is also fully working on early access systems for Aurora, with Intel OneAPI DPC++/SYCL builds part of its regular CI process (Note that due to limited hardware availability these tests target NVIDIA GPUs, tests on Intel based systems are run ad-hoc). RAJA is in the process of making SYCL fully work as a backend, but there are a number of outstanding issues which the team is currently working on.

Performance evaluations are ongoing in collaboration with application teams and the vendors. Numerous issues have already been identified and are actively worked on. Some of these issues can be worked around inside the Kokkos and RAJA layers (e.g. work around non-optimal heuristics in the compiler), others will require compiler fixes.

Next Steps The highest priority for both RAJA and Kokkos is now the further maturing of the backends for Aurora and Frontier, as well as optimization work guided by application teams. Besides our general support for application teams, we have chosen a few driver projects to focus the optimization efforts.

Addressing latency limitations in current application design is another critical topic. Many codes are now reaching a point on the production platforms, where kernel launch, memory transfer, and communication latencies are limiting factors. The Kokkos team is exploring concepts such as predefined kernel graphs as well as global arrays style communication to address these issues. Initial prototypes are available now, and need to be tested by applications.

Further work on the shared facilities for RAJA and Kokkos is ongoing.

4.1.16 WBS 2.3.1.19 Argo: Low-Level Resource Management for the OS and Runtime

The goal of the Argo project [65] is to augment and optimize existing OS/R components for use in production HPC systems, providing portable, open source, integrated software that improves the performance and scalability of and that offers increased functionality to exascale applications and runtime systems.

System resources should be managed in cooperation with applications and runtime systems to provide improved performance and resilience. This is motivated by the increasing complexity of HPC hardware and application software, which needs to be matched by corresponding increases in the capabilities of system management solutions.

The Argo software is developed as a toolbox—a collection of autonomous components that can be freely mixed and matched to best meet the user’s needs.

Project activities focus around four products:

1. AML: a library providing explicit, application-aware memory management for deep memory systems
2. UMap: a user level library incorporating NVRAM into complex memory hierarchy using a high performance `mmap`-like interface
3. PowerStack: power management infrastructure for optimizing performance of exascale applications under power or energy constraints
4. NRM: a daemon to centralize node management activities such as job management, resource management, and power management

AML

Overview AML is a memory management library designed to ease the use of complex memory topologies and complex data layout optimizations for HPC applications.

AML is a framework providing locality-preserving abstractions to application developers. In particular, AML aims to expose flexible interfaces to describe and reason about how applications deal with data layout, tiling of data, placement of data across hardware topologies, and affinity between work and data.

Key Challenges Between non-uniform memory access (NUMA) to regular DRAM, the 3-D stacked high-bandwidth memory, and the memory local to the accelerator devices such as GPUs, the increasing depth of the memory hierarchy presents exascale applications with a critical challenge of how to use the available heterogeneous memory resources effectively.

Standardized interfaces to manage complex memory hierarchies are lacking, and vendors are reluctant to innovate in this space in the absence of clear directions from the community. Coming up with an interface that is sufficiently expressive to cover the emerging and projected hardware advances, yet is simple enough and practical to be both acceptable and useful to the applications is the key challenge that we are working on addressing.

Solution Strategy AML provides explicit, application-aware memory management for deep memory systems. It offers a collection of building blocks that are generic, customizable, and composable. Applications can specialize the implementation of each offered abstraction and can mix and match the components as needed. AML building blocks can be used to create, for example, mechanisms to replicate data across memory devices. We provide some high-level interfaces, built on top of these building blocks, as examples of what can be done with AML and as features more readily usable by applications.

We provide applications and runtimes with a descriptive API for data access, where all data placement decisions are explicit, and so is the movement of data between different memory types. At the same time, the API does abstract the memory topology and other device complexities. We focus on optimizing data locality for current and future hardware generations; applications provide insights for static allocations, and we can also dynamically and asynchronously move or transform data to optimize for a particular device or to best take advantage of the memory hierarchy.

AML components are built on top of hardware drivers and system libraries (libnuma, hwloc, accelerator libraries, other ECP products). Figure 34 depicts the major components of AML, including:

- Placement: mechanisms to decide where data lives,
- Layout: descriptions of the shape of applications data,
- Movement: how to move data across devices (including transformations),
- Tiling: how to iterate over and generate collections of data chunks,
- High level abstractions and helpers.

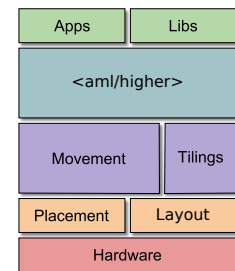


Figure 34: AML components.

Recent Progress This year AML contributions are distributed between new features and support for new devices.

As part of our effort to offer AML as a framework to abstract over many different types of heterogeneous memory devices, we implemented support for two more vendor APIs: OpenCL and Intel oneAPI (through Level Zero). This support allows application users to use AML on more platforms, while still offering the same level of portability as with the rest of the AML code: even though the initialization of a specific implementation of a building block is unique to that implementation, the code that makes use of this building block is generic across implementations. This design choice should keep application modifications at a reasonable level.

As we extend support for new devices in AML, the core abstractions offered by the library might end up being challenged by small differences in the low-level driver interfaces we support. This year, these challenges resulted in data movement facilities in AML receiving major improvements, in particular with respect to creating batches of requests to move small chunks of data. AML can now avoid creating request objects when many requests need to be performed at once, lowering the overhead of these types of movement.

Furthermore, we developed for our ExaSMR collaborators a new feature inspired by OpenMP 5 custom mappers. An OpenMP mapper makes it possible to describe which parts of a structure should be mapped onto accelerator devices. As it is a recent addition to the standard, it is currently poorly supported by compilers and lacks many features. For example, its current design is limited in its ability to filter or reorganize memory during the mapping process. We are using our tiling abstraction to build a more flexible interface for such

a feature. Finally, we recently introduced in the library a new facility to create custom allocators, on top of our placement building block (areas). These custom allocators offer a similar interface to the standard malloc/free C APIs, but allow for customization of the algorithm used to manage chunks of free memory. For example, we offer allocators that do not release memory to the devices on `free`, instead keeping it for future allocation requests, or allocators that always use the same chunk size for all requests. This new allocator facility should improve the performance and flexibility of applications with specific use cases (e.g., many temporary allocations).

Preliminary Experiences on Early Access Systems The latest AML release (0.2.0) includes support for Intel oneAPI and AMD HIP, which enables its use across CPUs and GPUs of Early Access Systems at ALCF and OLCF. We have successfully compiled it on the Spock system (Frontier EAS) at OLCF and we verified that the testsuite completed successfully.

Next Steps In a past collaboration with the author of the XSBench application, we demonstrated a consistent speedup across a variety of shared memory systems using the AML replicaset abstraction. As depicted in Figure 35, we were able to outperform or match existing solutions involving OpenMP or MPI by using this abstraction. Since the previous iteration of this report, we also implemented the mapper abstraction described above. We are planning to continue this collaboration to implement a versatile replicaset abstraction using the aforementioned mapper in order to duplicate complex data structures in low-latency memories. The goal is to accelerate OpenMC application while minimizing changes in the original code.

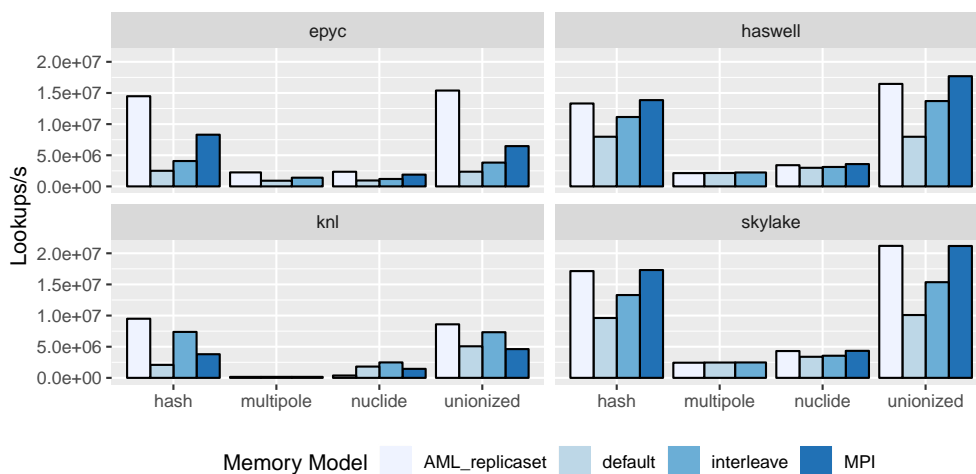


Figure 35: Performance of AML replicaset on different hardware architectures

We will continue to improve on our mapper facility based on the feedback from the ExaSMR developers. In particular, we are looking to split this feature into two interacting but independent components: a facility to describe how collections of complex application data structures are organized in memory, and a facility to navigate such data structures to transform and map them into a separate virtual memory region. Our goal is to improve the flexibility of our mapper to allow for more optimizations in the number of data movements and data allocations when data is moved between devices. A flexible enough mapper should also allow us to perform an optimization similar to the replicaset: replicating subsets of the memory being mapped on different memories to improve application performance.

UMap

Overview UMap is a user level library providing a high performance mmap-like interface that can be tuned to application needs without requiring OS customization or impacting system configuration. UMap enables applications to interact with out-of-core data sets as if in memory, and to configure parameters such as page buffer size and page size on a per-application basis.

Leadership supercomputers feature a diversity of storage, from node-local persistent memory and NVMe SSDs to network-interconnected flash memory and HDD. Interacting with large persistent data sources is critical to exascale applications that harness the power of data analytics and machine learning. The UMap user-level library enables user-space page management of data located in the memory/storage hierarchy. By providing a memory map interface, applications can interact with large data sets as if in memory. As a user level library, a UMap page fault handler can be easily adapted to access patterns in applications and to storage characteristics, reducing latency and improving performance.

Key Challenges As ECP applications transition to include ML and data analytics as integral components of workflows, persistent memories and low latency storage devices offer new alternatives to hold portions of very large global data sets within the fabric of the computing system. These new applications drive new access patterns, i.e. read-dominated analysis of observational or simulation data rather than write-mostly checkpoints. The combination of new technologies (byte addressable, very low latency, asymmetric read/write latency), new insertion points (node local, Top of Rack or other intermediate storage, global FS, external distributed storage servers), and new applications (in-situ analytics, experimental + simulation data analysis, ML batched data sets) present challenges both to the traditional memory/storage dichotomy as well as to traditional HPC I/O libraries tailored to checkpoint transmission.

Solution Strategy We prioritize four design choices for UMap based on surveying realistic use cases. First, we choose to implement UMap as a user-level library so that it can maintain compatibility with the fast-moving Linux kernel without the need to track and modify for frequent kernel updates. Also, we employ the recent `userfaultfd` mechanism, rather than the signal handling + callback function approach to reduce overhead and performance variance in multi-threaded applications. Third, we target an adaptive solution that sustains performance even at high concurrency for data-intensive applications, which often employ a large number of threads for hiding data access latency. Our design pays particular consideration on load imbalance among service threads to improve the utilization of shared resources even when data accesses to pages are skewed. UMap dynamically balances workloads among all service threads to eliminate bottleneck on serving hot pages. Finally, for flexible and portable tuning on different computing systems, UMap provides both API and environmental controls to enable configurable page sizes, eviction strategy, application-specific prefetching, and detailed diagnosis information to the programmer. The UMap software architecture is shown in Figure 36.

Recent Progress In recent months, we have released MP-UMap, providing multi-process support for the UMap facility. MP-UMap enables multiple processes on a compute node to share the UMap page buffer so that pages accessed from a mapped file are accessible to MPI ranks on the same node. MP-UMap is implemented as a collection of libraries that allows multiple processes to share a single file-backed UMap buffer. Just like UMap, it uses user-space page fault handler based on the `userfaultfd` Linux feature (starting with 4.3 linux kernel). The use case is to have a multi-process application accessing a large file through cached pages, i.e. out-of-core execution using memory map. Presently, `userfaultfd` does not support Write-Protection(WP) of shared memory, which limits MP-UMap to read-only applications. However, we have identified an experimental Linux kernel <https://github.com/xzpeter/linux/tree/uffd-wp-shmem-hugetlbfs-v4> where WP capability is being actively developed. Our feasibility investigation has shown that this experimental version will help us in upgrading MP-UMap's capability to support read-write applications. Thus, going forward we will require capability of running such experimental Linux versions on EAS platforms to test/use advanced capabilities like MP-UMap. Ultimately, as has happened previously, we have helped push adoption of the experimental feature into the mainstream linux distribution.

The LLNL UMap team demonstrated graph analysis algorithms based on graph traversal that use the UMap handler. By integrating with SICM's persistent memory C++ allocator called Metall into UMap, the UMap handler provides memory mapped access to very large graphs dynamically allocated and stored in binary form in one or more file(s). The algorithm suite consists of a dynamic graph construction benchmark, breadth-first search, connected components, and the ExaGraph miniVite benchmark. Combined use of UMap and Metall require the write-protect feature of `userfaultfd`, which is available in the linux 5.10 kernel and beyond. Currently this version is not available on EAS machines. Having EAS nodes with newer kernel

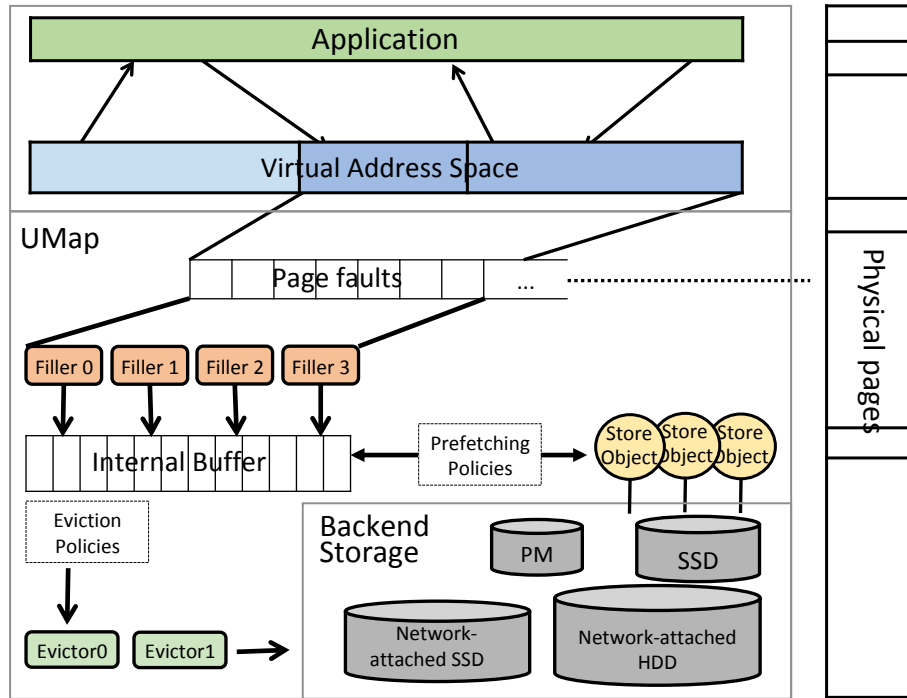


Figure 36: UMap Handler architecture

versions would help with testing and CI of Metall+UMap enabled ExaGraph applications that have been demonstrated.

An adaptive buffer management scheme was added to UMap to monitor and react at runtime to adjust the page buffer size. With this optimization, the page buffer can grow and shrink to reflect memory space availability changes.

A paper on UMap was published in IEEE Transactions on Parallel and Distributed Systems: Peng, I.B., Gokhale, M., Youssef, K., Iwabuchi, K. and Pearce, R., 2021. Enabling Scalable and Extensible Memory-mapped Datastores in Userspace. doi: 10.1109/TPDS.2021.3086302

EAS Experiences UMap and MP-UMap have been demonstrated on Frontier architecture EAS systems. No porting was required as UMap in read-only mode runs on the standard system software stack. Specifically, UMap was run on a node of Redwood, an El Capitan COE machine.

UMap-MP was run Spock as documented in the STPR19-19 Milestone report. Functionality of MP-UMap APIs were tested with a Spock Compute Node by using test apps and scripts available at <https://github.com/LLNL/umap/tree/mpumap/tests/umap-service>.

The full UMap and MP-UMap functionality cannot be run on EAS systems until the Linux kernel is upgraded to 5.10 for UMap and 5.14 for MP-UMap.

Next Steps In the coming year, we plan to continue outreach to application teams within ECP and in the science/data community. We are in collaboration with the Umpire team and are integrating UMap with Umpire, a new capability to be delivered in FY22. UMap with Umpire will give applications a common API to manage multiple sorts of memories including persistent memory through UMap and main CPU and GPU memory resources. The Exagraph collaboration is in progress. Selected Exagraph algorithms can memory map persistent data such as binary format graphs and intermediate data structures through UMap using the SparseStore and Metall.

Recognizing that new forms of memory will be available for Exascale architectures, we are enhancing the core UMap handler to exploit a multi-tier memory hierarchy in which pages can migrate between capacity/latency tiers.

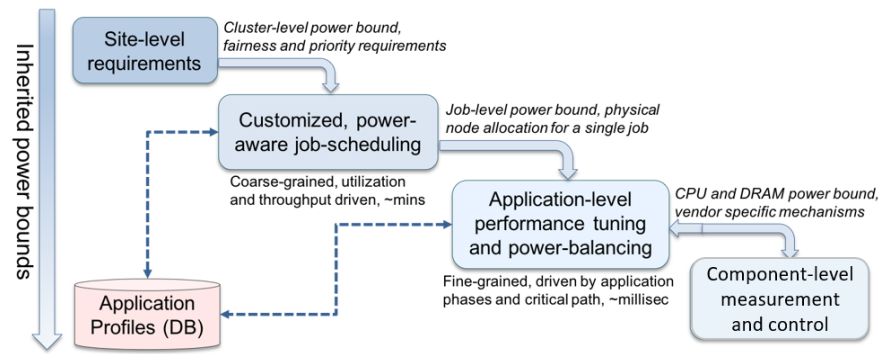


Figure 37: Envisioned PowerStack

PowerStack

Overview Power remains a critical constraint for Exascale. As we design supercomputers with higher heterogeneity at larger scales, power becomes an expensive and limited resource. Inefficient management of power leads to added operational costs as well as low scientific throughput. Although hardware advances will contribute a certain amount towards achieving high energy efficiency, several vendors agree that these will not be sufficient in isolation—creating a need for a sophisticated system software approach. Significant advances in software technologies are thus required to ensure that Exascale systems achieve high performance with effective utilization of available power. Distributing available power to nodes (and components such as GPUs) while adhering to system, job and node constraints involves complex decision making in software.

The ECP PowerStack sub-area in Argo explores hierarchical interfaces for power management at three specific levels: batch job schedulers, job-level runtime systems, and node-level managers. Each level will provide options for adaptive management depending on requirements of the supercomputing site under consideration. Site-specific requirements such as cluster-level power bounds, user fairness, or job priorities will be translated as inputs to the job scheduler. The job scheduler will choose power-aware scheduling plugins to ensure compliance, with the primary responsibility being management of allocations across multiple users and diverse workloads. Such allocations (physical nodes and job-level power bounds) will serve as inputs to a fine-grained, job-level runtime system to manage specific application ranks, in-turn relying on vendor-agnostic node-level measurement and control mechanisms. Figure 37 presents an overview of the envisioned PowerStack, which takes a holistic approach to power management. Additionally, power management support for science workflows (e.g., MuMMI Cancer workflow, E3SM climate models), in-situ visualization, and workflow management infrastructures (e.g., Kokkos, Caliper) is being developed. Interfaces with ATDM projects such as LLNL’s Flux are also being developed. Furthermore, solutions for co-scheduling challenges for extremely heterogeneous architectures are being designed as a part of a university subcontract to University of Arizona.

This project is essential for ECP because it enables power management of Exascale applications and science workflows on modern heterogeneous architectures, where optimal performance often depends on how resources are scheduled efficiently across power domains (e.g., GPUs, or co-scheduling). The project is also essential to allow for better throughput and utilization of such heterogeneous clusters, and for allowing applications to operate safely and optimally with power and energy constraints when needed. This project is also essential for building a sophisticated hierarchical software stack proposed by the ECP ATDM (LLNL) and Flux projects, as well as community standardization efforts such as the PowerAPI standard. Additionally, the project fulfills an essential need for ECP by enabling vendor and academic collaborations that provide for accelerated adoption of best practices and better interoperability at scale. By leveraging the software developed in this project, compute centers can safely operate under power and energy constraints while maximizing performance and scientific throughput.

Key Challenges Power management in software is challenging due to the dynamic phase behavior of applications, processor manufacturing variability, and the increasing heterogeneity of node-level components. While several scattered research efforts exist, a majority of these efforts are site-specific, require substantial

programmer effort, and often result in suboptimal application performance and system throughput. Additionally, these approaches are not production-ready and are not designed to cooperate in an integrated manner. A holistic, generalizable and extensible approach is still missing in the HPC community, and a goal for the ECP Argo PowerSteering project is to provide a solution for this technology gap.

Another set of challenges come from portability issues. Existing solutions are targeted toward specific microarchitectures (typically Intel) as well as specific programming models (typically MPI-only and traditional benchmarks). Additionally, some of the existing solutions violate the specified power budget before reaching a steady state, resulting in power fluctuations as well as unsafe operation. As part of this project, we strive to provide portability across multiple platforms (IBM, NVIDIA, ARM, AMD, etc), multiple programming models that enable workflows (through Kokkos or Caliper, or specific science workflow studies such as E3SM or MuMMI). Such portability and support of vendor-neutrality is important for safe operation using both hardware-level and application-level information for adaptive configuration selection and critical path analysis.

Solution Strategy As discussed earlier, our solution is to develop an end-to-end deployable stack, that combines coarse-grained power-scheduling (Flux, SLURM) with fine-grained job-level runtime system (Intel GEOPM) while ensuring vendor neutrality through node-level interfaces in Variorum. Such a stack can typically operate transparently to user applications. At the scheduler level, we are working on extending SLURM and Flux resource managers to be power-aware. Here, we are looking at both static, coarse-grained power management and variation-aware scheduling in Flux, as well as portability through SLURM SPANK plugins. For the job-level, a power management runtime system called GEOPM that will optimize performance of Exascale scientific applications transparently is being developed in collaboration with Intel. At the node-level, vendor-neutral interfaces are being developed as part of Variorum library, to allow for support for Intel, IBM, AMD, ARM, and HPE/Cray platforms. In order to accomplish portability and smooth integration across domains, we are closely collaborating with ECP MuMMI workflow project, the E3SM workflow project, ECP Flux, Kokkos and Caliper, and with the University of Arizona. We are actively engaging ECP users in order to support power management in a non-intrusive and transparent manner.

Recent Progress We achieved three milestones through FY21 in September 2021. In the first milestone, delivered in January 2021, the LLNL power team added the AMD and ARM architectures to Variorum. Variorum is an extensible library for exposing monitor and control capabilities of low-level hardware knobs. Variorum provides vendor-neutral APIs such that the user can query or control hardware knobs without needing to know the underlying vendor's implementation. These APIs enable HPC application developers to gain a better understanding of performance through various metrics for the devices contained within the node. Additionally, the APIs may enable system software to control hardware knobs to optimize for a particular goal. Details about Variorum can be found at <https://variorum.readthedocs.io/en/latest/>. Additionally, JSON-based interfaces that allow Variorum to interact with runtime systems and schedulers were also developed and hardened in this milestone, allowing support for interaction with ECP system software such as Kokkos, Caliper, and Flux. The team also hardened the codebase further to include continuous integration, additional documentation, concrete examples and tests.

In the second milestone in FY21, delivered in May 2021, we integrated GEOPM and NRM (Argo's Node Resource Manager) with a co-launch mechanism. We introduced a new wrapper script named `geopmnrmlaunch` along with its support components to launch GEOPM alongside NRM. The script supports the launch of GEOPM in two modes: the process mode and the thread mode. The goal was to compartmentalize power management decisions as follows: (1) manage job-level power bound and extract job-level power efficiency through GEOPM, and (2) manage node-level resource limiting and optimization goals through NRM. This design provides the basis for several advantages over the legacy PowerStack design. For example, hierarchical assignment of power optimization goals is now possible from the job scheduler to the NRM because GEOPM integrates with the job scheduler (SLURM). Furthermore, since NRM supports several containerization technologies out of the box, GEOPM can indirectly support containerized workflows with a limitation that power-assignment is specified at power domain boundaries. Additionally, compartmentalization of the power optimization goals enables separating job-level goals from node-level goals. This separation enables us to define level-specific goals, such as improving the time spent on the critical path (Instructions per Second or IPS) at the job level or improving power efficiency at the node level (IPS per Watt or IPS/W).

In the final milestone in FY21, delivered in Sept 2021, PowerStack team at LLNL and University of Arizona worked on three parallel goals of (1) Designing co-scheduling policies for ECP applications and comparing different policies for throughput (average turnaround times), (2) Integrating Variorum with Caliper in order to enable several Caliper workflows to leverage Variorum, and (3) Extending Variorum JSON API to include power domain queries. By leveraging initial codes and data from FY20, we conducted a detailed analysis of co-scheduling applications using a real HPC trace from LLNL’s Cab cluster. This included an analysis of which applications can be co-scheduled effectively to allow for space and time sharing of resources, and how the trade-off space between node utilization, throughput and individual application performance can be modeled. A scheduling simulator was designed and implemented for the same. We used NAS benchmarks as well as E3SM for co-scheduling studies. The Caliper service for Variorum and the JSON API for power domain queries further strengthened the codebase to allow for integration with higher-level system software.

Additionally, LLNL continues to work with a multi-vendor team towards a CRADA involving Intel, HPE, ARM and IBM – industry partners that are helping us drive vendor-neutral solutions to power management; and are actively engaged in the community effort for PowerStack homogenization. We are also working in collaboration with PowerAPI team for the same. We established the PowerStack community charter in June 2018, involving collaborators across multiple vendors (Intel, IBM, ARM, HPE, AMD, NVIDIA, Cray), academic institutions (TU Munich, Univ. Tokyo, Univ. Bologna, Univ. Arizona), and national laboratories (Argonne National Lab). The goal for this team is to design a holistic, flexible and extensible concept of a software stack ecosystem for power management. Over the past 3.5 years, this group is looking at engineering and research challenges, along with RFP/procurement designs through active vendor interaction and involvement. A joint BoF with the PowerAPI team, titled “Community-Driven Efforts for Energy Efficiency in HPC Software Stack,” is being held at SC21 (<https://sc21.supercomputing.org/presentation/?id=bof163&sess=sess404>).

Preliminary Experiences on Early Access Systems The PowerStack team is working closely with AMD for co-designing and enabling low-level power knobs from the user space with Variorum for both CPUs and GPUs as part of the El Capitan statement of work. An AMD CPU port will be included as part of the Variorum February 2022 release, and the AMD GPU port for Variorum is under development (as some of the hardware supported features are still being sketched out). These ports include both telemetry and control, and are currently being tested on internal AMD nodes. This is because specialized kernel drivers and root privileges are required for such testing and evaluation of power management knobs. We are able to build and expose both telemetry and control, as well as run comparison studies, as described in detail in the report for our February 2022 milestone. Once fully integrated, Variorum will support both Frontier and El Capitan architectures, and will also support power-aware scheduling through Flux.

Next Steps We will continue our research and development work as planned toward the FY22 milestones. More specifically, we will continue development for variorum library to allow better support for AMD CPUs, AMD GPUs and other architectures, along with better JSON APIs for system software integration with Kokkos, Caliper and Flux on El Capitan. We will continue to extend Intel GEOPM’s new codebase, progress with advanced integration of GEOPM and NRM, continue development of scheduler components such as Flux and SLURM, work on GPU power capping research, and enable user-space access to power management on diverse architectures. We will expand our collaborations for science workflows, such as MuMMI and E2SM, including support for Caliper and Kokkos power management. We will also continue to further explore co-scheduling challenges in power management (University of Arizona) and multi-tenancy issues in power management on heterogenous architectures, and lead the efforts on multi-vendor CRADA. Additionally, we will forge the pathway for converged computing platforms to adopt the power stack by collaborating on HPC+Cloud initiatives.

NRM

Overview Argo Node Resource Manager (NRM) is a daemon running on the compute nodes. It centralizes node management activities such as local application launching, resource management, and power management.

NRM interacts with both global resource management services (e.g., job scheduler) and with application components and runtime services running on the node. It acts as a control infrastructure to enable custom resource management policies at the node level. Applications can influence these mechanisms, both directly

(through explicit API calls used, e.g., to request additional resources on the node) and indirectly (by having their run-time behavior monitored by NRM).

Key Challenges Many ECP applications have a complex runtime structure, ranging from in situ data analysis, through an ensemble of largely independent individual subjobs, to arbitrarily complex workflow structures. At the same time, HPC hardware complexity increases as well, from deeper memory hierarchies to heterogeneous compute resources and performance fluctuating based on power/thermal constraints.

Even in the common case of each compute node being allocated exclusively to a single job, managing available node resources can be a challenge. If a compute node is shared among multiple job components (a likely scenario considering the reduced cost of data transfers), these components—if allowed to freely share node resources—could interfere with one another, resulting in suboptimal performance. It is the NRM’s job to rein in this complexity by acting as a coarse-grained resource arbitrator.

Solution Strategy NRM uses an active approach to resource management. Physical and logical resources on the compute nodes are configured, discovered, and accounted for. NRM can manage compute nodes and individual components of parallel workloads, in the sense that it performs an active accounting of two types of interfaces for those workloads: sensors and actuators. These abstracted components are available through an upstream API which enables their discovery and management.

The NRM daemon supports power actuators, as well as sensors based on CPU counters and power information. We also provide a simple API that application processes can use to periodically update the NRM on their progress. This gives NRM reliable feedback on the efficacy of its power policies, and it can also be used for a more robust identification of the critical path, rather than relying on heuristics based on performance counters.

In addition to those capabilities, NRM’s configuration format allows for the configuration of actuators and active polling sensors based on arbitrary executables. Figure 38 shows a reading of NRM’s sensor for two RAPL sensors along with CPU counter instrumentation.

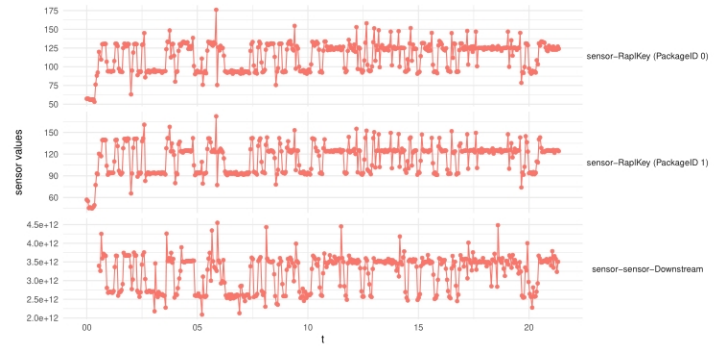


Figure 38: NRM sensor output.

In order to provide these capabilities, the NRM daemon manages the launching, management, and stopping of parallel workloads through a unified interface. Resources can be dynamically reconfigured at run time; these interfaces are provided for use from applications and from global services.

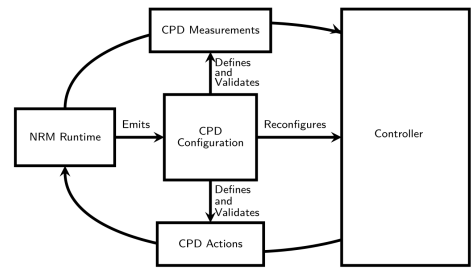


Figure 39: Control Problem Description (CPD).

Other than sensor and actuator accounting capabilities, NRM provides optional support for autonomic node management. This support is provided through the accounting of node-local autonomic goals and constraints, which are expressed in terms of available nodes and sensors. These goals and constraints can be inspected by the users, global services (GRM), and applications through NRM’s unified interface. Inspection is achieved through the emission of a Control Problem Description (CPD), which outlines active sensors, actuators, goals, and constraints. This scheme is outlined in Figure 39.

Recent Progress The NRM daemon continues to undergo major improvements in its interfaces to interact with the rest of the system. This includes improvements to the upstream API to allow for a passthrough mode, where NRM only manages the collection of events from sensors and discovery and management of actuators but does not enforce any control policy by itself. This passthrough mode was used successfully to explore alternative

control schemes inspired by autonomic computing and a first integration with GEOPM (acting as the global control loop across nodes). These efforts also led to the improvement of the python interface offered by NRM.

We continue to improve our support for various sensors expected to be of value on exascale systems, with the inclusion of an OMPT interface to monitor OpenMP activity in controlled applications similar to our existing PMPI support.

Our support for actuators is also growing, with the support for launching an arbitrary command as an actuator, including the ability of the command to receive information (pid) about an application as arguments.

We continue supporting the `libnrm` C library that can be linked to applications in order to provide reports on application progress to NRM. We have recently improved this interface to increase the resolution of the timestamps associated with events, allowing for nanosecond precision in the monitoring of fast events.

Preliminary Experiences on Early Access Systems We developed the Argo power management glue layer for Intel discrete GPUs (APMIDG), which can communicate with NRM to manage the power consumption of Intel discrete GPUs. It allows us to simultaneously measure the power consumption, current frequency, and temperature of Intel discrete GPUs on different domains (if available on target hardware). It also allows us to control the power capping and the frequency range. We verified its power management capabilities on Intel Arctic Sound GPUs, available as Aurora Early Access Systems. In addition, we will verify the capabilities of Intel Ponte Vecchio GPUs (Aurora GPUs) as soon as they become available. The source code of APMIDG is maintained at <https://github.com/anlsys/apmidg>, which contains only public information.

Next Steps We are working on significantly improving the handling of sensor readings (events) within the NRM. This includes gathering more metadata about the physical location (on the topology) and scope (logic section of code/thread/rank) that sensors report information about. We are also looking into a more flexible mechanism to handle smoothing/integration of sensor data across time: indeed, a critical component of any control loop is its ability to understand how the system behaved since the last control action. We are investigating possible designs to allow users to configure how such integration should be done, as it currently only allows for averaging data across time windows.

We are also working on slightly modifying the NRM infrastructure to allow for it to be installed from Spack, delegating some of its privileged operations to external tools (i.e., no admin rights needed anymore in the core infrastructure).

We are planning to expand the list of resources managed by NRM by adding support for other vendor-specific mechanisms as well.

4.2 WBS 2.3.2 DEVELOPMENT TOOLS

End State: A suite of compilers and development tools aimed at improving developer productivity across increasingly complex heterogeneous architectures, primarily focused on those architectures expected for the upcoming Exascale platforms of Frontier and Aurora.

4.2.1 *Scope and Requirements*

For Exascale systems, the compilers, profilers, debuggers, and other software development tools must be increasingly sophisticated to give software developers insight into the behavior of not only the application and the underlying hardware but also the details corresponding to the underlying programming model implementation and supporting runtimes (e.g., capturing details of locality and affinity). These capabilities should be enhanced with further integration into the supporting compiler infrastructure and lower layers of the system software stack (e.g., threading, runtime systems, and data transport libraries), and hardware support. Most of the infrastructure will be released as open source, as many of them already are, with a supplementary goal of transferring the technology into commercial products (including reuse by vendors of ECP enhancements to LLVM, such as Fortran/Flang, or direct distributions by vendors of software on platforms). Given the diversity of Exascale systems architectures, some subset of the tools may be specific to one or more architectural features and is potentially best implemented and supported by the vendor; however,

the vendor will be encouraged to use open APIs to provide portability, additional innovation, and integration into the tool suite and the overall software stack.

4.2.2 Assumptions and Feasibility

The overarching goal of improving developer productivity for Exascale platforms introduces new issues of scale that will require more lightweight methods, hierarchical approaches, and improved techniques to guide the developer in understanding the characteristics of their applications and to discover sources of the errors and performance issues. Additional efforts for both static and dynamic analysis tools to help identify lurking bugs in a program, such as race conditions, are also likely needed. The suite of needed capabilities spans interfaces to hardware-centric resources (e.g., hardware counters, interconnects, and memory hierarchies) to a scalable infrastructure that can collect, organize, and distill data to help identify performance bottlenecks and transform them into an actionable set of steps and information for the software developer. Therefore, these tools share significant challenges due to the increase in data and the resulting issues with management, storage, selection, analysis, and interactive data exploration. This increased data volume stems from multiple sources, including increased concurrency, processor counts, additional hardware sensors and counters on the systems, and increasing complexity in application codes and workflows.

Compilers obviously play a fundamental role in the overall programming environment but can also serve as a powerful entry point for the overall tool infrastructure. In addition to optimizations and performance profiling, compiler-based tools can help with aspects of correctness, establishing connections between programming model implementations and the underlying runtime infrastructures, and auto-tuning. In many cases, today's compiler infrastructure is proprietary and closed source, limiting the amount of flexibility for integration and exploration into the Exascale development environment. In addition to vendor compiler options, this project aims to provide an open source compiler capability (via the LLVM ecosystem) that can play a role in better supporting and addressing the challenges of programming at Exascale.

4.2.3 Objectives

This project will design, develop, and deploy an Exascale suite of compilers and development tools for development, analysis, and optimization of applications, libraries, and infrastructure from the programming environments of the project. The project will seek to leverage techniques for common and identified problem patterns and create new techniques for data exploration related to profiling and debugging and support advanced techniques such as autotuning and compiler integration. We will leverage the open-source LLVM compiler ecosystem. For tools, the overarching goal is to leverage and integrate the data measurement, acquisition, storage, and analysis and visualization techniques being developed in other projects of the software stack. These efforts will require collaboration and integration with system monitoring and various layers within the software stack.

4.2.4 Plan

Multiple projects will be supported under this effort. To ensure relevance to DOE missions, a majority of these efforts shall be DOE laboratory led, and collaborate with existing activities within the broader HPC community. Initial efforts will focus on identifying the core capabilities needed by the selected ECP applications, components of the software stack, expected hardware features, and the selected industry activities from within the Hardware and Integration focus area. The supported projects will target and implement early versions of their software on both CORAL and APEX systems, with an ultimate target of production-ready deployment on Frontier and Aurora. Throughout this effort the, applications teams and other elements of the software stack will evaluate and provide feedback on their functionality, performance, and robustness. These goals will be evaluated annually at the ST reviews and for milestones.

4.2.5 Risk and Mitigation Strategies

A primary risk exists in terms of adoption of the various tools by the broader community, including support by system vendors. Past experience has shown that a combination of laboratory-supported open source software and vendor-optimized solutions built around standard APIs that encourage innovation across multiple

platforms is a viable approach, and this will be undertaken. We will track this risk primarily via the risk register.

Given its wide use within a range of different communities, and its modular design principles, the project’s open source compiler activities will focus on the use of the LLVM compiler ecosystem (<https://llvm.org/>) as a path to reduce both scope and complexity risks and leverage with an already established path for NRE investments across multiple vendors. The compilers and their effectiveness are tracked in the risk register. In fact, in 2020, we created a fork of the llvm-project upstream repository (see <https://github.com/llvm-doe-org>) to capture, integrate, and test LLVM projects. This repo also serves as a risk mitigation option that can be easily enabled if other compilers are not working successfully on the target platforms.

Another major risk for projects in this area is the lack of low-level access to hardware and software necessary for using emerging architectural features. Many of these nascent architectural features have immature implementations and software interfaces that must be refined prior to release to the broader community. This project should be at the forefront of this interaction with early delivery systems. This risk is also tracked in the risk register for compilers, which are particularly vulnerable.

4.2.6 *Future Trends*

Future architectures are becoming more heterogeneous and complex [66]. As such, the role of languages, compilers, runtime systems, and performance and debugging tools will become increasingly important for productivity and performance portability. In particular, our ECP strategy focuses on improving the open source LLVM compiler and runtime ecosystem; LLVM has gained considerable traction in the vendor software community, and it is the core of many existing heterogeneous compiler systems from NVIDIA, AMD, Intel, ARM, IBM, and others. Over the past two years, several vendors including IBM and Intel have abandoned their proprietary compiler software in favor of LLVM. We foresee that this trend will continue, which is why we have organized the Development Tools technical area around LLVM-oriented projects. We expect for many of our contributions to LLVM to address these trends for the entire community and will persist long after ECP ends. For example, our contributions for directive-based features for heterogeneous computing (e.g., OpenMP, OpenACC) will not only provide direct capabilities to ECP applications, but it will also impact the redesign and optimization of the LLVM infrastructure to support heterogeneous computing. In a second example, Flang (open source Fortran compiler for LLVM; [the second version is also known as F18]) will become increasingly important to the worldwide Fortran application base, as vendors find it easier to maintain and deploy to their own Fortran frontend (based on Flang). Furthermore, as Flang become increasingly robust, researchers and vendors developing new architectures will have immediate access to Flang, making initial Fortran support straightforward in ways similar to what we are seeing in Clang as the community C/C++ frontend.

4.2.7 *WBS 2.3.2.01 Development Tools Software Development Kits*

Overview The Software Development Tools SDK is a collection of independent projects specifically targeted to address performance analysis at scale. The primary responsibility of the SDK is to coordinate the disparate development, testing, and deployment activities of many individual projects to produce a unified set of tools ready for use on the upcoming exascale machines. The efforts in support of the SDK are designed to fit within the overarching goal to leverage and integrate data measurement, acquisition, storage, analysis, and visualization techniques being developed across the ECP ST ecosystem.

Key Challenges In addition to the general challenges faced by all of the SDKs outlined in Section 4.5.7, the unique position of the Development Tools SDK between the hardware teams and the application developers requires additional effort in preparing today’s software to run on yet-unknown architectures and runtimes to be delivered by the end of ECP.

Solution Strategy The primary mechanism for mitigating risk in the SDK is the readiness survey. This survey is designed to assess the current status of each product in the SDK in six key areas: software availability, documentation, testing, Spack build support, SDK integration, and path forward technology utilization. By

periodically assessing the progress of the individual L4 products in the SDK, we will use the survey to identify and resolve current hardware architecture dependencies, plan for future architecture changes, and increase adoption of the continuous integration (CI) testing workflow to reduce this risk.

Critically, the survey will allow us to accomplish this by providing a direct communication channel between the SDK maintainers and the L4 product developers allowing us to identify current architecture dependencies in each project and compare them with existing and emerging ECP platforms. Our initial efforts will be to increase support for today's heterogeneous CPU architectures across the DOE facilities (e.g., x86, Power, ARM, etc.) to ensure a minimum level of usability on these platforms. We will then focus on current accelerator architectures- namely GPGPU computing. As new architectures arise, we will re-issue the survey and use this same process to provide guidance to the L4 product as they develop support for them.

The survey also allows us to monitor the increased adoption of the proposed ECP CI testing workflow. This will be crucial to understanding each project's interoperability with not only the other projects within the Tools SDK, but all applications across the ECP Software Technologies landscape. Additionally, it will serve as a bridge between the HI teams working with the facilities and the software teams working across the SDK. By relaying new hardware requirements from the facilities to the software developers, we can closely monitor support for both new and existing systems. Conversely, giving feedback to the facilities regarding compiler support and buildability of library dependencies will guide software adoption on those platforms.

Recent Progress The Readiness Survey was re-issued to each L4 product in September 2021. There have been no changes to planned GPU support. Work is actively ongoing to get the tools operating with the Intel GPUs for Aurora via the Arcticus early access system and the AMD GPUs for Frontier via the Spock early access system. In Q321, NERSC enabled ECP users to evaluate their preliminary version of Perlmutter. Only two of the tools L4 projects are targeting work for that system, currently. We expect that number to grow in the next few months.

Support for automated testing remains a challenge area that all of the projects are aware of and will continue to be a point of focus for the SDK. Thus far, two of the six products, Dyninst and TAU, have had successful runs of their complete test suites on pre-exascale systems. With this work, both products now have working tests that can be employed through scriptable executions. This represents an essential component of software sustainability to demonstrate and track correctness in the presence of code changes for these products. Additionally, all of the L4 projects have both an entry in the E4S test suite as well as 'smoke test' functionality in their Spack packages. These testing modalities will be automatically executed by the E4S team when assessing releases. The Spack smoke tests are also executed by the Spack CI pipeline when evaluating pull requests for a package as well as part of preparation for a Spack release.

Continuous Integration (CI) testing remains a still-larger challenge for the SDK. This is due in part to some products not having scriptable testing capabilities and also in part to more general challenges of using CI at the facilities. With the decommissioning of the OSTI Gitlab, we have moved to a distributed model of running CI at each facility separately. We currently have a preliminary CI pipeline that can be manually executed to build the entire SDK.

Starting in Q122, the SDK will be expanded from six to thirteen L4 projects. This was the result of a request from LLNL affiliates to improve support and utilization of their internally-developed tools. These changes were introduced too late to be explicitly included in the FY22 P6 activities, but they will be incorporated into the SDK's milestone efforts, nonetheless.

The Tools SDK has been an active contributor to developing the E4S Community Policies (Figure 11). From the FY21 readiness survey results, two of the L4 projects are planning to be member packages in E4S. They are working toward compatibility in FY22 and beyond. It should be noted that these policies are not a requirement for inclusion in the SDK. Instead, they are good-faith efforts to demonstrate how software in the sciences can be made more accessible and portable.

Next Steps In FY21, we continued our efforts to assess builds with multiple compilers on early access systems. Results from these tests were fed back into the L4 products to guide development of spack packages, bug/issue-reporting workflows, and integration into the greater ECP software ecosystem. As the number of early access systems increases in FY22, we expect this work to remain central to our efforts. Additionally, we want more of the L4 members to be consistently built under the auspices of the SDK development efforts as an addition to the various builds carried out by the Spack and E4S developers. A large portion of our SDK

work allocation in FY22 will be dedicated to this task with an emphasis on utilizing Continuous Integration at the various labs as a vehicle.

4.2.8 WBS 2.3.2.06 Exa-PAPI

Overview The Exa-PAPI project is developing new performance counter monitoring capabilities as well as power management support for novel and advanced ECP hardware, and software technologies. Exa-PAPI builds upon classic-PAPI functionality and strengthens its path to exascale with a standard interface and methodology for using low-level performance counters in CPUs, GPUs, on/off-chip memory, interconnects, and the I/O system, including energy/power management.

In addition to providing hardware counter-based information, a standardizing layer for monitoring software-defined events (SDE) is being incorporated that exposes the internal behavior of runtime systems and libraries, such as communication and math libraries, to the applications. As a result, the notion of performance events is broadened from strictly hardware-related events to include software-based information. Enabling monitoring of both hardware and software events provides more flexibility to scientific application developers when capturing performance information.

Key Challenges Widely deployed and widely used, PAPI has established itself as fundamental software infrastructure in every application domain where improving performance can be mission critical. However, processor and system designs have been experiencing radical changes. Systems now combine multi-core CPUs and accelerators, shared and distributed memory, PCI-express and other interconnects, and power efficiency is emerging as a primary design constraint. These changes pose new challenges and bring new opportunities to PAPI. At the same time, the ever-increasing importance of communication and synchronization costs in parallel applications, as well as the emergence of task-based programming paradigms, pose challenges to the development of performance-critical applications and create a need for standardizing performance events that originate from various ECP software layers.

Solution Strategy The Exa-PAPI team is preparing PAPI support to stand up to the challenges posed by exascale systems by widening its applicability and providing robust support for exascale hardware resources; supporting finer-grain measurement and control of power, thus offering software developers a basic building block for dynamic application optimization under power constraints; extending PAPI to support software-defined events; and applying semantic analysis to hardware counters so that the application developer can better make sense of the ever-growing list of raw hardware performance events that can be measured during execution.

In summary, the team is channeling the monitoring capabilities of hardware counters, power usage, software-defined events into a robust PAPI software package.

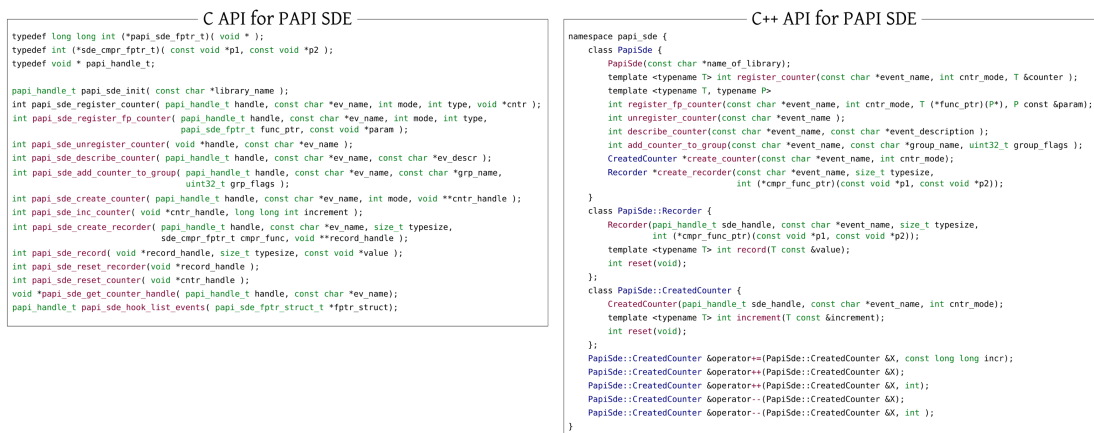


Figure 40: Existing PAPI SDE C API and new SDE C++ API.

Recent Progress The Exa-PAPI team is in the process to prepare a major release—PAPI 7.0.0—scheduled to be shipped in November 2021. For this release, the team has designed and developed a C++ API for utilizing PAPI’s software-defined events (SDEs) from within third-party libraries and applications that are written in C++. This is a major addition to the already implemented and released C and FORTRAN APIs for the PAPI SDE functionality (shipped with PAPI 6.0.0 in March 2020). The example code in Figure 40 demonstrate the C and C++ API for PAPI SDEs.

Furthermore, PAPI 7.0.0 will release newly developed support for the latest monitoring features on the Intel Gen9 GPUs. This was accomplished under NDA and in very close collaboration with the Intel team. The new PAPI capabilities for monitoring Intel Gen9 include GPU hardware events, and memory performance metrics (bytes read/written/transferred from/to L3). Monitoring of GPU and memory performance counters aids developers in producing more efficient code by profiling the utilization of the latest GPU resources and diagnosing performance bottlenecks. The objective of this development was to enable transparent access to the Gen9 performance and memory metrics via Intel’s OneAPI Level-Zero Interface. A unique feature of this development enables users to apply two different collection modes via the PAPI interface:

The Time-based Collection Mode Counter values are aggregated and stored in a buffer. Counter values can be read at any given time during the execution, which means performance counter monitoring is completely autonomous from the kernels running in the GPUs.

The Kernel-Based Collection Mode Counter monitoring is per kernel, and performance counter data is not available during kernel execution but only after the kernel finishes execution. In this mode, the Level-Zero API uses internal barriers, which implies that if two kernels overlap, then the execution of the second kernel is pushed out until the first kernel finishes.

Another major change in PAPI 7.0.0 is the refactoring of the PAPI CUDA component and new developments to support NVIDIA compute capability 7.0 and greater. This required the use of the CUDA/CUPTI 11 Profiling and Perfworks APIs. The PAPI CUDA component has been redesigned so that it works in equal measure for NVIDIA compute capabilities < 7.0 and ≥ 7.0 .

Last but not least, the team designed and implemented PAPI support for FUGAKU’s A64FX Arm architecture. This work includes the development of new Counter Analysis Toolkit (CAT) benchmarks and refinements of PAPI’s CAT data analysis; specifically, the extension of PAPI’s CAT with MPI and “distributed memory”-aware benchmarks and analysis in order to stress all cores/node. The CAT analysis of the A64FX raw hardware counters resulted in the definition of 36 new PAPI presets for FUGAKU users.

Next Steps Next steps for each effort have already been identified.

Modernize Atomics and Reduce Overhead in PAPI When PAPI calls are made from within different thread contexts, PAPI uses internal mechanisms for guaranteeing thread safety. Atomic operations have a central role in these mechanisms, since they introduce less overhead than mutexes. However, the support for atomic operations in the current PAPI framework is limited, and many opportunities are missed. Projects such as the atomic_ops library offer a much wider support for atomic operations, which the team plans to leverage in PAPI.

Improved Multithreading and OpenMP Support in PAPI Beyond the atomic operations for guaranteeing correctness, PAPI will modernize support for threads. PAPI was originally designed in a time when threads were exotic, and OpenMP was nonexistent. The Exa-PAPI team will improve the interoperability with thread systems and OpenMP in particular.

Development of PAPI Support for New ECP Hardware Development of new PAPI components, as well as new features in current PAPI components due to underlying vendor interface updates. Development of new PAPI presets (predefined events) for newly released hardware.

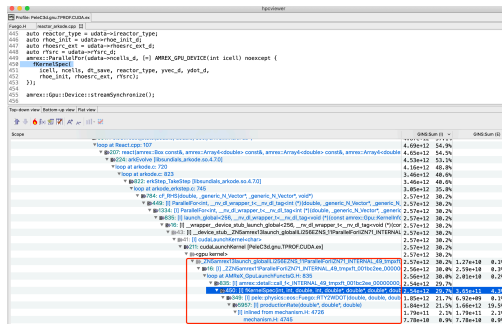
4.2.9 WBS 2.3.2.08 HPCToolkit

Overview The HPCToolkit project is working to develop performance measurement and analysis tools to help ECP application, library, runtime, and tool developers understand where and why their software does not fully exploit hardware resources within and across nodes of extreme-scale parallel systems. Key deliverables of the project are a suite of software tools that developers need to measure and analyze the performance of parallel software as it executes on existing ECP testbeds and new technologies needed to measure and analyze performance on forthcoming GPU-accelerated exascale systems.

To provide a foundation for performance measurement and analysis, the project team is working with community stakeholders, including standards committees, vendors, and open source developers to improve hardware and software support for measurement and attribution of application performance on extreme-scale parallel systems. The project team has been engaging vendors to improve hardware support for performance measurement in next-generation GPUs and working with other software teams to design and integrate new capabilities into operating systems, runtime systems, communication libraries, and application frameworks that will enhance the ability of software tools to accurately measure and attribute code performance on extreme-scale parallel systems. Using emerging hardware and software interfaces for monitoring code performance on both CPUs and GPUs, the project team is working to extend capabilities to measure and analyze computation, data movement, communication, and I/O as a program executes to pinpoint scalability bottlenecks, quantify resource consumption, and assess inefficiencies.

Key Challenges Today's fastest supercomputers and forthcoming exascale systems all employ GPU-accelerated compute nodes. Almost all of the computational power of GPU-accelerated compute nodes comes from GPUs rather than CPUs. GPU-accelerated compute nodes have complex memory hierarchies that include multiple memory technologies with different bandwidth and latency characteristics. In addition, GPU-accelerated compute nodes have non-uniform connections between memories and computational elements (CPUs and GPUs). Furthermore, three emerging DOE supercomputers (Perlmutter, Aurora, and Frontier) will feature GPUs from different vendors (NVIDIA, Intel, and AMD). There are significant differences in the underlying organization of these GPUs as well as their hardware and software support for performance measurement. For performance tools, the need to support multiple CPU and GPU architectures significantly increases tool complexity. At the same time, the complexity of applications is increasing dramatically as developers struggle to expose billion-way parallelism, map computation onto heterogeneous computing elements, and cope with the growing complexity of memory hierarchies. While application developers can employ abstractions to hide some of the complexity of emerging parallel systems, performance tools must be intimately familiar with each of the features added to these systems to improve performance or efficiency, develop measurement and analysis techniques that assess how well these features are being exploited, and then relate these measurements back to software to create actionable feedback that will guide developers to improve the performance, efficiency, and scalability of their applications.

Solution Strategy Development of HPCToolkit as part of ECP is focused on preparing it for production use at exascale by enhancing it in several ways. First, the team is adding new capabilities to measure and analyze interactions between software and key hardware subsystems in extreme-scale platforms, including GPUs and the complex memory hierarchies on GPU-accelerated compute nodes. A major focus of this effort is developing new capabilities for measurement and analysis of performance on GPUs. Second, the team is working to improve performance attribution given optimized code for a large collection of complex node-level programming models used by ECP developers, including vendor specific programming models such as CUDA, HIP, and Data Parallel C++, open source community programming models such as OpenMP, and template-based programming models developed at national laboratories such as LLNL's RAJA and Sandia's Kokkos. To support this effort, the project team is enhancing the Dyninst binary analysis toolkit, which is also used by other ECP tools. A major focus of this effort is to support analysis of GPU binaries. Third, the team is improving the scalability of HPCToolkit so that it can be used to measure and analyze extreme-scale executions. Fourth, the project team is working to improve the robustness of the tools across the range of architectures used as ECP platforms. Finally, the project team will work other ECP teams to ensure that they benefit from HPCToolkit's capabilities to measure, analyze, attribute, and diagnose performance issues on ECP testbeds and forthcoming exascale systems.



(a)



(b)

Figure 41: (a) HPCToolkit’s `hpcviewer` showing a detailed attribution of GPU performance metrics in a profile of PeleC. (b) HPCToolkit’s `hpctraceviewer` showing CPU and GPU trace lines for LAMMPS.

Recent Progress

- The project team developed a unified GPU monitoring substrate. Upon this substrate, they developed support for collecting summary metrics for kernel launches, memory copies, and synchronizations on AMD, Intel, and NVIDIA GPUs.
- The project team enhanced support for fine-grained measurement of GPU computations using PC sampling on NVIDIA GPUs and binary instrumentation on Intel GPUs. HPCToolkit uses instruction-level measurements to reconstruct calling context trees and parses NVIDIA’s extended line map to reconstruct GPU inlining call chains to help developers understand performance of complex GPU kernels. Figure 41(a) shows a calling context for PeleC, an ECP application for adaptive-mesh compressible hydrodynamics code for reacting flows; the highlighted region shows a reconstruction of GPU calling contexts with multiple inlined GPU device functions.
- The project team enhanced HPCToolkit for collecting and visualizing traces of both CPU and GPU activity. Figure 41(b) shows a trace of a GPU-accelerated execution of LAMMPS, an ECP application for parallel molecular dynamics simulation, using 512 NVIDIA A100 GPUs. Each GPU kernel is related to the CPU calling context in which it was launched, which helps developers understand the performance of complex applications.
- The project team developed a novel approach that utilizes both shared and distributed memory parallelism to analyze and aggregate sparse data representations of performance measurements from every rank, thread and GPU stream in a program execution.
- The project team improved the reliability of HPCToolkit’s `hpcrun` for monitoring complex dynamic library loading and unloading and interacting with the application when other software tools are present.

Preliminary Experiences on Early Access systems

HPCToolkit The project team is working to deliver tracing and profiling of GPU activity atop AMD’s `rocTracer` API and to assess GPU kernel activity using hardware counters using AMD’s `rocProfiler` API. HPCToolkit builds and runs on the Spock and Crusher early access systems. At present, the project team is working closely with AMD to resolve issues measuring GPU-accelerated applications with the `rocTracer` and `rocProfiler` APIs by providing test cases that highlight issues of concern.

In addition, the project team has been collaborating with AMD to develop support for monitoring the performance of OpenMP offload in the runtime for AMD’s AOMP compiler. The code is integrated into AMD’s development branch and has been submitted upstream for integration into LLVM OpenMP. A development branch of HPCToolkit uses this support to attribute GPU activity to OpenMP `target` regions. The HPCToolkit project team is working to integrate this emerging capability into a forthcoming release.

Dyninst Dyninst builds and runs on the Spock and Crusher early access systems. HPCToolkit uses Dyninst to analyze AMD CPU binaries to attribute performance measurements to source code contexts, including functions, loops, and lines.

Next Steps Next steps for each effort have already been identified. The team will integrate GPU measurement and analysis capabilities using hardware counters. Solutions for fine-grained computation measurement on Intel GPUs will be improved, and the team will explore new solutions for fine-grained measurement on AMD GPUs. Efforts will continue to refine the reliability and performance of monitoring operations on dynamic libraries. The team will work with the open-source community to upstream the team’s GPU measurement support into the community version of the `libomptarget` offloading library. Finally, we will continue to work with DOE and platform vendors to evaluate and refine software interfaces for measuring GPU performance.

4.2.10 WBS 2.3.2.10 PROTEAS-TUNE: Programming Toolchain for Emerging Architectures and Systems

Key Challenges Programmer productivity and performance portability are two of the most important challenges facing users of future exascale computing platforms. Application developers targeting ECP architectures will find it increasingly difficult to meet these two challenges without integrated capabilities that allow for flexibility, composability, and interoperability across a mixture of programming, runtime, and architectural components.

Solution Strategy The PROTEAS-TUNE project was formed as a strategic response to this challenge. (The PROTEAS-TUNE project is the result of merger in FY20 of two previous ECP projects: PROTEAS [PROgramming Toolchain for Emerging Architectures and Systems] and Y-Tune: Autotuning for Cross-Architecture Optimization and Code Generation.) This project has three high-level goals. First, PROTEAS-TUNE will provide a programming pathway to anticipated exascale architectures by addressing programmability and portability concerns of emerging technology trends seen in emerging architectures. In particular, the project focuses on improvements to LLVM and OpenACC. Additionally, the team has significant experience with CUDA, OpenCL, and other programming models that will enable ECP applications teams to explore programming options to find the most effective and productive approaches without constraining programming models or software solutions. Second, PROTEAS-TUNE will prototype an integrated programming framework strategy will deliver solutions on these emerging architectures that will be further refined for these architectural capabilities, and make sure that they transition to vendors, standards activities, applications, and facilities. Thirdly, PROTEAS-TUNE includes autotuning which makes it possible to separate a high-level C/C++/FORTRAN implementation from architecture-specific implementation (OpenMP, OpenACC, CUDA, etc.), optimization, and tuning. It also provides a flexible programming framework and integrated toolchain that will provide ECP applications the opportunity to work with programming abstractions and to evaluate solutions that address the exascale programming challenges they face.

Specifically, the PROTEAS-TUNE focuses on several thrusts to improve capabilities and performance portability for applications on exascale architectures:

- Improve the core-LLVM compiler ecosystem
- Design and implement the OpenACC heterogeneous programming model for C/C++ in Clang/LLVM (Clacc)
- Design and implement the OpenACC heterogeneous programming model for Fortran in Flang/LLVM (Flacc)
- Use performance modeling and optimization to enable code transformation and performance portability
- Refine autotuning for OpenMP and OpenACC programming models to directly target challenges with heterogeneous architectures
- Improve performance measurement and analysis tools (TAU) for the target exascale architectures and apply it to applications to improve performance

- Develop and implement portable software abstractions (Papyrus) for managing persistent memory
- Contribute to the SYCL programming model by developing a representative benchmark suite and evaluating SYCL implementation readiness across relevant architectures
- Aggressively engage applications, SDK, vendor, and software teams for demonstration and deployment
- In collaboration with SOLLVE and Flang, develop a DOE ECP fork of LLVM that will be the clearinghouse for ECP modifications of LLVM (see <https://github.com/llvm-doe-org>)
- Conduct benchmarking and development of SYCL implementations across ECP platforms

Importantly, the team’s solutions are based on significant, continuing work with LLVM, OpenACC, OpenMP, ARES HLIR, OpenARC, TAU, SuRF and CHiLL. The team has extensive experience and a demonstrated track record of accomplishment in all aspects of this proposed work including existing software deployments, interaction with application teams, vendor interaction, and participation in open source community and standards organizations. Also, the team champions its successful solutions in ECP procurements, community standards, and open-source software stacks, like LLVM, in order to improve their use.

Recent Progress Our recent work has focused on OpenACC and Clacc, OpenACC and Flacc, TAU, LLVM, SYCL, and outreach and collaboration.

OpenACC and Clacc [67] Develop production-quality, standard-conforming OpenACC compiler and runtime support as an extension of Clang/LLVM. See Section 4.2.12.

OpenACC and Flacc Develop production-quality, standard-conforming OpenACC compiler and runtime support as an extension of Flang/LLVM.

TAU Expand performance analysis with TAU by adding additional functionality for new architectures. Improve a widely-used performance analysis framework by adding functionality for new architectures and software systems. See Section 4.2.17.

LLVM In collaboration with numerous other ECP projects, PROTEAS is contributing improvements to the LLVM compiler infrastructure. These improvements include simple bugfixes to the existing infrastructure, monitoring Flang progress, developing Clacc (see Section 4.2.12), developing Flacc (see Section 4.2.14), and developing a DOE ECP fork of LLVM for our work.

SYCL Our SYCL evaluation identified significant differences in the performance of various SYCL implementations, and we continue to improve our SYCL benchmark suite. See Section 4.2.19.

Outreach and Collaboration with ECP Applications Teams We have interacted with over a dozen applications teams to help prepare their applications for ECP. See Sections 4.2.12, 4.2.18, and 4.2.17.

Next Steps Our next efforts for Clacc, Flacc, TAU, LLVM, and SYCL are as follows:

Clacc Continue developing OpenACC support by lowering OpenACC directives to use the existing LLVM OpenMP infrastructure.

Flacc Continue developing OpenACC support by finishing the development of the OpenACC dialect for MLIR and beginning to develop the runtime system on the existing LLVM OpenMP infrastructure.

TAU Improve performance instrumentation for deep memory hierarchies in TAU, focusing primarily on various GPUs and emerging NVM.

ECP LLVM Fork Improve support for continuous integration of the ECP LLVM fork.

SYCL Evaluation Continue to improve our SYCL benchmark suite and work with researchers and vendors to enhance SYCL options for ECP users. See Section 4.2.19.

Preliminary Experiences on Early Access systems Across our tasks, we have some preliminary experiences on the ECP Early Access Systems (EAS) as detailed below.

LLVM LLVM has been build and used successfully on Spock and Crusher. An automatic build of a recent LLVM version for Spock, e.g., weekly, is planned. We also intend to do that for Crusher. LLVM (including our extensions) work as expected. Existing AMD GPU bugs, especially but not exclusively when using OpenMP offload, prevented full applications from running. Proxy applications successfully compiled and run.

Clacc The Clacc project is not using the EASs.

LLVM-DOE The LLVM-DOE fork does not use the EASs for continuous integration testing or development.

Flacc The Flacc project is not using the EASs.

Autotuning The ytopt autotuning framework has not yet been tested on the EAS Systems.

Bricks The Brick Library has been used to generate performance data on spock, and has been tested on crusher.

TAU TAU has been ported to the most recent EAS systems. On the OLCF system Crusher, TAU has been tested with the most recent software and firmware updates, up to and including ROCm 4.5.2. TAU is being used to help tune the XGC application from the ECP WDMApp project on Crusher. On the ALCF system Arcticus, TAU has been tested up to and including the most recent versions of OneAPI. On the NERSC system Perlmutter, TAU has been ported and tested with the NVIDIA A100 devices and most recent versions of CUDA.

Papyrus The Papyrus project is not using the EASs.

SYCL The SYCL project is not using the EASs.

4.2.11 WBS 2.3.2.10 PROTEAS-TUNE: LLVM

Overview LLVM, winner of the 2012 ACM Software System Award, has become an integral part of the software-development ecosystem for optimizing compilers, dynamic-language execution engines, source-code analysis and transformation tools, debuggers and linkers, and a whole host of programming-language and tool chain-related components. Now heavily used in both academia and industry, where it allows for rapid development of production-quality tools, LLVM is increasingly used in work targeted at HPC. LLVM components are integral parts of the programming environments on our upcoming Exascale systems, and smaller-scale systems as well, being not only popular open-source dependencies, but are critical parts of the commercial tool chains provided by essentially all relevant vendors.

Key Challenges LLVM is well suited to the compilation of code from C++ and other languages on CPU hardware, and for some models, GPU hardware, but lacks the kind of high-level optimizations necessary to enable performance-portable programming across future architectures.

- LLVM lacks the ability to understand and optimize parallelism constructs within parallel programs.
- LLVM lacks the ability to perform high-level loop transformations to take advantage of complex memory hierarchies and parallel-execution capabilities.

Without these abilities, code compiled well for LLVM must be presented to the compiler in a form already tuned for a specific architecture, including expressions of parallelism suited for the particular characteristics of the target machine. It is, however, unfeasible to tune our entire workload of applications in this way for multiple target architectures. Autotuning helps this problem by allowing dynamic analysis to supplement static cost modeling, which is always fundamentally limited, but without the ability to perform complex transformations, both the parallel and serial execution speed of the resulting programs will be suboptimal.

There are two remaining challenges that we are addressing: The first is that deploying autotuning relying on source-to-source transformations is difficult because maintaining these separate source kernel versions is

practically difficult. The second is that, as a general matter, performance improvements can be obtained by specializing code and runtime as opposed to limiting ourselves to ahead-of-time code generation.

Solution Strategy We are developing two significant enhancements to LLVM’s core infrastructure, and many other LLVM components. These enhancements are grouped into two categories:

1. Enhancements to LLVM’s inter-procedural analysis, and an improved representation of parallelism constructs, to allow LLVM to propagate information across boundaries otherwise imposed by parallelism constructs, and to allow LLVM to transform the parallelism constructs themselves.
2. Enhancements to LLVM’s loop-optimization infrastructure to allow the direction of a sequence of loop transformations to loop nests, exposing these features to users through Clang pragmas (in addition to being available at an API level to tools such as autotuners), enabling those transformations to execute as specified, and otherwise enhancing the loop-optimization infrastructure.

As part of this project, we’re investigating both fundamental IR level enhancements (as part of the Kitsune development), as well as runtime call aware optimizations that deal with the classical lowering of parallelism into runtime calls. The latter mechanism is being implemented upstream as an OpenMP optimization pass, while the Kitsune work is, at present, more exploratory.

To address autotuning and the need for code specialization, we are developing a just-in-time compilation technology with integrates naturally with the C++ language as well as embedding of (domain specific) languages into C/C++ programs.

Recent Progress For parallelism, we have implemented several new features in upstream LLVM including an OpenMP-aware optimization pass that performs various optimizations specific to OpenMP code on the host and device (=GPU) [68]. It is run by default with medium and high optimizations enabled (-O2 and -O3). In addition to transformations it will provide user feedback in form of optimization remarks (-Rpass=openmp-opt)

Extension to the Attributor inter-procedural optimization framework that transparently applies transformations across the boundary between sequential and parallel code. This upstream work will reduce the overhead parallelism introduces due to missed classical optimizations [69].

We prototyped heterogeneous LLVM-IR modules which allow host and device code to coexist in the same LLVM-IR file and therefore be optimized with a holistic view. Our approach was already discussed with the community and needs to be further refined and tested.

For loop optimizations, we have implemented several new features in LLVM and Clang, including the OpenMP 5.1 `tile` directive and the `unroll` directive [70]. We also continued development of the OpenMPIRBuilder which refactors Clang’s OpenMP lowering such that is can also be used from MLIR, specifically Flang. The OpenMPIRBuilder now also supports OpenMP 5.1 loop transformations and Clang can make use of either of the two lowering paths [71].

To facilitate autotuning (ref. Section 4.2.15), we further enhanced the semantics-preserving implementation of loop transformations using Polly [72]. Additional supported loop transformations include array packing, unroll, unroll-and-jam, loop distribution and fusion.

We have developed a prototype C++ compiler, based on Clang, supporting an extension that enables programmers to embed (domain specific) languages inside their C-like programs [73]. That is, we allow a new type of Clang plugin to bridge the gap between classical code, e.g., C or C++, and code written in a different language, e.g., a quantum or tensor domain specific language (DSL). The latter two examples have been successfully prototyped as well.

In support of the Clacc project 4.2.12, we developed a compiler pass from a subset of OpenACC to Tapir, building on the Kitsune project. This enables optimizations and alternative backends to the primary compilation path of Clacc.

All our efforts have also been featured in many talks, tutorials, and so on at LLVM developers’ meetings over the last couple of years.

Next Steps We will continue to prototype implementations, discuss them with the LLVM community, and then refine them for integration in upstream LLVM.

For the C++ JIT technology, we will also continue to pursue standardization at the C++ standards committee.

In addition, we are implementing autotuning technology based on the loop transformation improvements, and other improvements developed by this project. This will enable an easy-to-use autotuning capability for applications on Exascale systems.

Parallelism specific optimizations will further be improved through Attributor enhancements upstream and more capabilities for the OpenMP-aware aware optimization pass. Generalization of the latter to other parallel models is planned as well.

Building on existing and in-progress MLIR efforts, e.g., FLACC 4.2.14 and Kitsune, we will develop Tapir backends for multiple parallel MLIR dialects, including the OpenACC MLIR dialect. This should show the advantages of multiple levels of parallelism awareness in compiler representations.

To enable optimizations across the host-device boundary we are continuing to work on heterogeneous LLVM-IR modules in order to integrate them into upstream LLVM.

4.2.12 WBS 2.3.2.10 PROTEAS-TUNE - Clacc: OpenACC in Clang and LLVM

Overview Heterogeneous and manycore processors (e.g., multicore CPUs, GPUs, Xeon Phi, etc.) are becoming the de facto architectures for current HPC platforms and future Exascale platforms. These architectures are drastically diverse in functionality, performance, programmability, and scalability, significantly increasing the complexity that ECP app developers face as they attempt to fully utilize available hardware.

A key enabling technology pursued as part of PROTEAS-TUNE is OpenACC. While OpenMP has historically focused on shared-memory multi-core, OpenACC was launched in 2010 as a portable programming model for heterogeneous accelerators. Championed by institutions like NVIDIA, PGI, and ORNL, OpenACC has evolved into one of the most portable and well recognized programming models for accelerators today.

Despite the importance of OpenACC, the only non-academic open-source OpenACC compiler cited by the OpenACC website is GCC [74]. However, GCC has lagged behind commercial compilers, such as NVIDIA's, in providing production-quality support for the latest OpenACC specifications [75]. Moreover, GCC is known within the compiler community to be challenging to extend and, especially within the DOE, is losing favor to Clang and LLVM for new compiler research and development efforts.

Clacc [67] is a major component of PROTEAS-TUNE. Overall, the goal is to build on Clang and LLVM to develop an open-source, production-quality OpenACC compiler ecosystem that is easily extensible and that utilizes the latest research in compiler technology. Such an ecosystem is critical to the successful acceleration of ECP applications using modern HPC hardware. The PROTEAS-TUNE objectives for Clacc are as follows:

1. Develop production-quality, standard-conforming OpenACC compiler, runtime, and profiling support as an extension of Clang and LLVM. Two compilation modes are being developed: (a) traditional mode, which produces a binary, and (b) source-to-source mode, which produces OpenMP source.
2. As part of the design, leverage the Clang ecosystem to enable the future construction of source-level OpenACC tools, such as pretty printers, analyzers, lint tools, debugger extensions, and editor extensions.
3. Throughout development, actively contribute improvements to the OpenACC specification, and actively contribute mutually beneficial improvements to the upstream Clang and LLVM infrastructure.
4. As the work matures, contribute OpenACC support itself to upstream Clang and LLVM so that it can be used by the broader HPC and parallel programming communities.

Key Challenges

OpenACC Support Developing production-quality, standards-conforming OpenACC compiler and runtime support is a large undertaking. Complicating that undertaking further is the need for optimization strategies that are competitive with existing commercial compilers, such as NVIDIA's, which have been developed over many years since before the conception of the OpenACC standard.

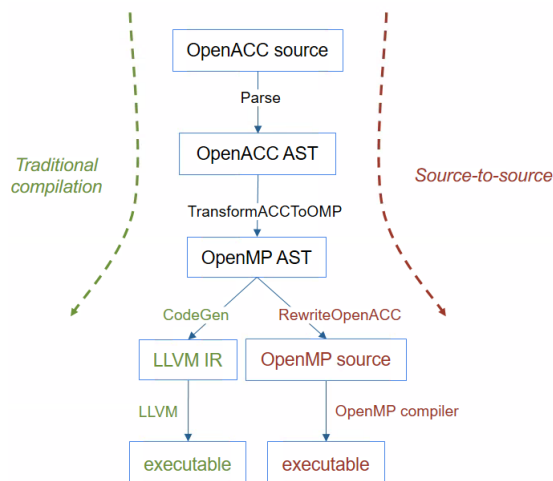


Figure 42: Clacc workflow.

Source-to-Source Lowering OpenACC to OpenMP significantly reduces the effort to implement OpenACC and offers additional capabilities, such as OpenACC support for proprietary OpenMP compilers. However, not all OpenACC features are actually representable in OpenMP. Moreover, there are several challenges from Clang’s AST, the source-level representation. First, it was designed to be immutable. Second, the AST represents the source after preprocessor expansions, which harm readability and can prevent compilation with other compilers. Third, sophisticated analyses and optimizations are critical for lowering OpenACC’s descriptive language to the more prescriptive language of OpenMP, but these are best implemented at the level of LLVM IR not the Clang AST.

Production-Quality Clang and LLVM are sophisticated tools with a complex codebase and a large team of developers who diligently screen contributions to maintain a clean design and correct operation. As for any production-quality compiler, developing and contributing improvements to Clang and LLVM can be significantly more challenging and time-consuming than for research-quality compilers.

OpenMP Alternative We believe that OpenACC’s current momentum as the go-to directive-based language for accelerators will continue into the foreseeable future. Nevertheless, some potential OpenACC adopters hesitate over concerns that OpenACC will one day be replaced by OpenMP features. A tool to migrate OpenACC applications to OpenMP could alleviate such concerns, encourage adoption of OpenACC, and thus advance utilization of acceleration hardware in ECP applications.

Solution Strategy

1. A key Clacc design feature is lowering OpenACC to OpenMP. Benefits are as follow:
 - (a) By building on Clang and LLVM’s OpenMP support, it reduces the effort necessary to construct a production-quality OpenACC implementation.
 - (b) It enables OpenACC support on OpenMP compilers other than Clang, including proprietary compilers.
 - (c) It facilitates repurposing existing OpenMP static analysis and debugging tools for the sake of OpenACC.
 - (d) It facilitates porting applications from OpenACC to OpenMP to alleviate the aforementioned concerns about developing applications in OpenACC.
3. To handle Clang’s immutable AST, Clacc’s design includes a TransformACCToOMP component (Figure 42), which reuses Clang’s TreeTransform feature, which was originally designed for C++ template specialization.

4. To avoid preprocessor expansions in source-to-source mode, Clacc includes a RewriteOpenACC component, which reuses a Clang feature called Rewrite.
5. To represent OpenACC features not covered by the OpenMP standard, Clacc is developing original OpenMP extensions. To utilize LLVM IR analyses and optimizations, Clacc first translates OpenACC to our extended OpenMP, which Clacc then translates to LLVM IR.
6. To stage our development effort, we are initially implementing Clacc with two simplifications: we are implementing a prescriptive OpenACC interpretation for correct behavior, and we are implementing for C. We will then extend Clacc with necessary analyses for a descriptive interpretation and for C++.
7. Throughout Clacc development, we are continuously integrating the latest upstream Clang and LLVM changes, and we are running and extending the Clang and LLVM test suites to detect regressions and incompatibilities. We are also investigating OpenACC benchmarks [76] and validation test suites [75] to ensure correct OpenACC behavior and good performance.

Recent Progress

1. Extended Clacc to support additional OpenACC features, most notably the OpenACC runtime library routines, but also various directives, clauses, environment variables, and associated profiling interface functionality. Developed various OpenMP extensions to support this work.
2. Extended Clacc's OpenACC support to AMD GPU offload, and extended the LLVM DOE Fork's CI infrastructure to test Clacc on relevant platforms, including AMD GPUs and OLCF's Ascent.
3. Contributed to Clang and LLVM, including LLVM testing infrastructure support and original OpenMP extensions required for OpenACC support. Contributed to the OpenACC and OpenMP specifications.
4. Collaborated with the OpenACC organization to invite broader participation in OpenACC LLVM development. Efforts include docs for the LLVM DOE Fork to guide potential collaborators, co-mentoring a Google Summer of Code student, and organizing an OpenACC in LLVM mini-workshop.

Next Steps

1. Continue to implement Clacc support for critical OpenACC features based on the needs of ECP and other HPC apps and benchmarks, and pursue OpenACC optimizations and C++ support.
2. Continue contributions to upstream Clang and LLVM and to the OpenACC and OpenMP specifications.

4.2.13 WBS 2.3.2.10 PROTEAS-TUNE - LLVM-DOE: Creating and Maintaining a DOE Fork of LLVM

Overview The ECP funds multiple projects that develop compiler technologies, based on the popular, open-source LLVM compiler infrastructure project. This ecosystem allows customization to meet the unique needs of ECP, and a level of well-established mechanisms to deploy technologies through vendors and at DOE's leadership facilities. Importantly, this provides an alternative open-source compiler ecosystem to those provided by the vendor, thus reducing the dependence on the vendor's compilers, timelines, and staff (Risk 10032 that ST product will not function or meet performance targets).

In addition, most today's vendors already rely on LLVM as the foundation for their compiler ecosystems. This means ECP technology has a path back to vendors via LLVM itself or through a DOE-/ECP-focused fork of LLVM's open source repository. This work will focus on deployment to reduce Risk 10020.

More broadly, there are eight LLVM-related projects supported by ECP that have a risk of not being used if developers cannot easily access their contributions. This fork of LLVM will provide an opportunity for these projects to work collectively on establishing synergies, interoperability, address the unique needs of ECP, and mechanisms for making contributions back into the mainstream LLVM code base. The tasks to setup the DOE Fork of LLVM are as follows:

1. Set up a fork of the llvm-project upstream repository (see <https://github.com/llvm-doe-org>).

2. Enable continuous integration for the fork on various hardware of interests.
3. Enable LLVM ECP related projects to be able to push and test branches.
4. Setup status information for the continuous information results.

Solution Strategy The DOE LLVM repository is set up on GitHub as a fork of the llvm-project main repository also hosted on GitHub. This makes it easier to have a seamless synchronization with the main repository and keep all the GitHub main-fork integrated features. The GitHub repository is automatically mirrored in the GitLab premium instance hosted at ORNL. Furthermore, the continuous integration takes advantage of the GitLab CI infrastructure. This infrastructure is available on several machines from the ExCL lab as well as on Summit and Theta.

Recent Progress The fork has been set up with an automatic mirroring with the upstream repository. The mirroring is using GitHub Actions, and the fork is integrated with E4S releases.

Next Steps The team will add continuous integration on more hardware (i.e., most recent versions of Frontier and Aurora test nodes) and improve the test suite for the CI (e.g., OpenACC tests).

4.2.14 *WBS 2.3.2.10 PROTEAS-TUNE - FLACC and MLIR: Creating and Maintaining OpenACC in LLVM/Flang*

Overview Heterogeneous and manycore processors (e.g., multicore CPUs, GPUs, Xeon Phi) are becoming the de facto architectures for current HPC platforms and future Exascale platforms. These architectures are drastically diverse in functionality, performance, programmability, and scalability, significantly increasing the complexity that ECP app developers face as they attempt to fully utilize available hardware.

A key enabling technology pursued as part of PROTEAS is OpenACC. While OpenMP has historically focused on shared-memory multi-core, OpenACC was launched in 2010 as a portable programming model for heterogeneous accelerators. Championed by institutions like NVIDIA, PGI, and ORNL, OpenACC has evolved into one of the most portable and well recognized programming models for accelerators today.

Despite the importance of OpenACC, the only non-academic open-source OpenACC compiler cited by the OpenACC website is GCC [74]. However, GCC has lagged behind commercial compilers, such as PGI's, in providing production-quality support for the latest OpenACC specifications [75]. Moreover, GCC is known within the compiler community to be challenging to extend and, especially within the DOE, is losing favor to Clang and LLVM for new compiler research and development efforts.

Recent efforts to build a Fortran counter-part to Clang in LLVM project have been accelerated and big chunk of the Flang have been upstreamed. Directive-based programming model are heavily used in Fortran applications ported to accelerators. Unlike C and C++, Fortran does not have many alternatives.

FLACC proposes to develop a prototype OpenACC 3.0 implementation in Flang based on MLIR to fill this gap. As the implementation of the OpenMP target offload feature in Flang does not have a clear path, this work will help in this regard by sharing code in the MLIR dialects and lowering sections with the Flang and SOLLVE projects.

Key Challenges

OpenACC Support Developing production-quality, standards-conforming OpenACC compiler and runtime support is a large undertaking. Complicating that undertaking further is the need for optimization strategies that are competitive with existing commercial compilers, such as PGI's, which have been developed over many years since before the conception of the OpenACC standard.

MLIR Flang and the OpenACC support for it rely on the MLIR project for the IR. MLIR has been upstreamed to the core LLVM project in early 2020 and it is still actively under development. Flang will be the first core frontend relying on MLIR.

Runtime LLVM does not include an OpenACC runtime but only one for OpenMP at the moment. This runtime can be generalized to support missing OpenACC features. This generalization needs to be accepted by the current OpenMP community.

OpenMP Stability As we plan to generalize the OpenMP runtime to support OpenACC, we will also rely on various part of the runtime that are already here. There has been some concern on the stability of the current OpenMP runtime implementation and especially the `textlibomptarget` responsible for the target offload part.

Solution Strategy

1. Flacc design follows a similar design as the OpenMP implementation for Flang. This design includes the following aspects:
 - (a) An OpenACC MLIR dialect part of the core MLIR project.
 - (b) A lowering from the Flang AST to a mix of FIR and OpenACC MLIR dialect.
 - (c) A progressive lowering from MLIR to LLVM IR with runtime call.

Recent Progress

1. OpenACC 3.0 parser is upstreamed to the Flang front-end. It covers the full specification and also implements the unparsing feature of Flang.
2. Semantic checking for OpenACC 3.0 is also upstreamed in Flang. While implementing this part, a new TableGen backed for directive-based language has been contributed upstreamed. This is used by OpenACC and OpenMP for both Clang and Flang.
3. Discussed Flacc at various ECP, HPC, and LLVM venues.

Next Steps

1. Continue the definition of the OpenACC MLIR dialect and complete the lowering to it.
2. Integration of Flacc runtime support with OpenMPIRBuilder being developed in the broader ECP Project.
3. Develop mapping of OpenACC runtime calls to OpenMP based on early work in Clacc.
4. Continue adding support to lower from MLIR to LLVM IR and runtime call.

4.2.15 WBS 2.3.2.10 PROTEAS-TUNE: Autotuning

Overview We are developing tools and an application development workflow that separates a high-level C/C++/FORTRAN implementation from an architecture-specific implementation (OpenMP, CUDA, etc.), optimization, and tuning. This approach will enable Exascale application developers to express and maintain a single, portable implementation of their computation that is also legal code that can be compiled and run by using standard tools. The autotuning compiler and search framework will transform the baseline code into a collection of highly-optimized implementations. This reduces the need for extensive manual tuning. Both code transformation and autotuning are essential in ECP for providing performance portability on Exascale platforms. Due to significant architectural differences in ECP platforms, attaining performance portability may require fundamentally different implementations of software – different strategies for parallelization, loop order, data layout, and exploiting SIMD/SIMT. A key concern of ECP is the high cost of developing and maintaining performance-portable applications for diverse Exascale architectures, including manycore CPUs and GPUs. Ideally Exascale application developers would express their computation separate from its mapping to hardware, while autotuning compilers can automate this mapping and achieve performance portability.

Key Challenges Autotuning has the potential to dramatically improve the performance portability of Petascale and Exascale applications. To date, autotuning has been used primarily in high-performance applications through tunable libraries or previously tuned application code that is integrated directly into the application. If autotuning is to be widely used in the HPC community, support for autotuning must address the software engineering challenges, manage configuration overheads, and continue to demonstrate significant performance gains and portability across architectures. In particular, tools that configure the application must be integrated into the application build process so that tuning can be reapplied as the application and target architectures evolve.

Solution Strategy We are developing pluggable software infrastructure that incorporates autotuning at different levels: compiler optimization, runtime configuration of application-level parameters and system software. To guarantee success in the ECP time frame, we are collaborating with application teams, such as SuperLU and QMCPACK, to impact performance of their codes and libraries.

The autotuning compiler strategy revolves CHiLL, which has the following distinguishing features: (1) Composable transformation and code generation, such that the same tool can be applied to multiple different application domains; (2) Extensible to new domain-specific transformations that can be represented as transformations on loop nest iteration spaces are also composable with existing transformations; (3) Optimization strategies and parameters exposed to autotuning: By exposing high-level expression of the autotuning search space as transformation recipes, the compiler writer, an expert programmer or embedded DSL designer can directly express how to compose transformations that lead to different implementations. A part of our efforts in ECP are to migrate these capabilities of CHiLL into the Clang/LLVM open-source compiler, as well as provide lightweight interfaces through Python, C++, and REST APIs/web services.

For example, we have developed a brick data layout library and code generator for stencil computations. Recent trends in computer architecture that favor computation over data movement incentivize high-order methods. Paradoxically, high-order codes can be challenging for compilers/optimization to attain high performance. Bricks enable high performance and make fine-grained data reuse and memory access information known at compile time. The SIMD code generation achieves performance portability for high-order stencils for both CPUs with wide SIMD units (Intel Knights Landing) and GPUs (NVIDIA Pascal). Integration with autotuning attains performance that is close to Roofline performance bound for both manycore CPU and GPU architectures and demonstrates strong scaling by reducing on-node data movement in communication.

ytopt/Surf is a machine-learning-based search software package for autotuning that consists of sampling a small number of input parameter configurations, evaluating them, and progressively fitting a surrogate model over the input-output space until exhausting the user-defined time or the maximum number of evaluations. The package is built based on Bayesian Optimization that solves any optimization problem and is especially useful when the objective function is difficult to evaluate. It provides an interface that deals with unconstrained and constrained problems. The software is designed to operate in the master-worker computational paradigm, where one master node fits the surrogate model and generates promising input configurations and worker nodes perform the computationally expensive evaluations and return the outputs to the master node. The asynchronous aspect of the search allows the search to avoid waiting for all the evaluation results before proceeding to the next iteration. As soon as an evaluation is finished, the data is used to retrain the surrogate model, which is then used to bias the search toward the promising configurations.

Recent Progress We have pursued the following activities this year.

Autotuning Capability in LLVM The key idea is to support the use of pragmas in the C++ source to guide transformations to be applied. These can include the types of transformation recipes used in CHiLL, but also parallelization directives for OpenMP and OpenACC that would interact with SOLLVE and PROTEAS. Our initial focus is the implementation of user/tool-directed optimizations in Polly, which is a polyhedral framework in LLVM with some similar features to CHiLL. Several existing open-source LLVM projects allowing for just-in-time (JIT) compilation of C++ code have been identified and are being evaluated for use with autotuning.

We have successfully used loop transformations directives implemented in Clang (ref. Section 4.2.11) to drive a machine learning process with ytopt. The correctness of these transformations are checked in the Clang compiler by Polly such that it is not possible to result in wrong results. The search strategies

used include a Globally Greedy [77], random search, beam search and our latest algorithm a customized Monte-Carlo Tree Search [72]. In addition to the PolyBench benchmark suite, we applied the search over a tree-shaped configuration space to three ECP Proxy apps, namely SW4Lite, CoMD and minrAMR with promising results.

ytopt/SuRF Supporting Autotuning Search Compiler directives such as pragmas can help programmers to separate an algorithm’s semantics from its optimization. Pragma directives for code transformations are useful for assisting program optimization and are already widely used in OpenMP. A prototype of user-directed loop transformations using Clang and Polly was implemented for the ECP SOLLVE project. Polly is LLVM’s polyhedral loop optimizer which makes it easy to apply specific transformations as directed by pragmas. While the SOLLVE team is working on integrating the changes into the official LLVM repository, only few of the changes have been upstreamed yet. The additional loop transformation directives supported are loop reversal (inverting the iteration order of a loop), loop interchange (permutating the order of nested loops), tiling, unroll(-and-jam), array packing (temporarily copying the data of a loop’s working set into a new buffer) and thread parallelization. More importantly, it supports composing multiple loop nest transformation in arbitrary order. Vectorization is also supported by LLVM’s dedicated loop vectorizer. These pragmas are intended to make applying common loop optimization technique much easier and allow better separation of a code’s semantics and its optimization.

We developed and extended ytopt autotuning framework that leverages Bayesian optimization to explore the parameter space search of LLVM Clang/Polly loop optimization pragmas. We integrated and compared four different supervised learning methods within Bayesian optimization and evaluated their effectiveness. We selected six of the most complex PolyBench benchmarks and apply the newly developed LLVM Clang/Polly loop optimization pragmas to the benchmarks to optimize them. We then used ytopt/SuRF framework to optimize the pragma parameters to improve their performance. The experimental results showed that our autotuning approach outperforms the other compiling methods to provide the smallest execution time for the benchmarks syr2k, 3mm, heat-3d, lu, and covariance with two large datasets in 200 code evaluations for effectively searching the parameter spaces with up to 170,368 different configurations.

In a number of settings, the search space exposed by the transformation pragmas in LLVM LLVM/Polly’s composable loop optimization is a tree, wherein each node represents a specific combination of loop transformations that can be applied to the code resulting from the parent node’s loop transformations. Consequently, Bayesian optimization within ytopt/SuRF will become ineffective. To that end, we have developed ytopt/MCTS, a search algorithm based on Monte Carlo tree search (MCTS). The algorithm consists of two phases: exploring loop transformations at different depths of the tree to identify promising regions in the tree search space and exploiting those regions by performing a local search. Moreover, a restart mechanism is used to avoid the MCTS getting trapped in a local solution. The best and worst solutions are transferred from the previous phases of the restarts to leverage the search history. We compared our approach with breadth-first, beam, global greedy, and random search methods using PolyBench kernels and ECP proxy applications. Experimental results showed that our MCTS algorithm finds pragma combinations with a speedup of 2.3x over Polly’s heuristic optimizations on average.

CCS is a autotuning interface that we developed recently for ECP Kokkos library. CCS aims at providing interoperability between autotuning frameworks and applications with autotuning needs. It does so by providing a C interface with which users can describe their tuning problem, but also write their own tuners, possibly in higher-level languages. At the start of the collaboration with Kokkos team, CCS did not match up exactly with the Kokkos model of tuning. In CCS, there were no input types, and the concept of output types, objective space in CCS, was much more complex, providing a system of constraints among those types. It also had a much more complex view of a good run, having an objective space concept to specify how well a given configuration performed. Through codesign, CCS has adopted the input type concept as the features space, while maintaining the full expressiveness of their other concepts. This is progressing to the point where, as users develop new tuners to the CCS infrastructure, they get the ability to tune Kokkos as well.

Brick Library We developed a code generator for the Brick Data Layout library for stencils that is performance-portable across CPU and GPU architectures, and addresses the needs of modern multi-stencil and high-order stencil computations. The key components of our approach that lead to performance portability are (1) a fine-grained brick data layout designed to exploit the inherent multidimensional spatial locality common to stencil computations; (2) vector code generation that can either target wide SIMD CPU instructions sets

such as AVX-512 and SIMT threads on GPUs; and, (3) integration with autotuning framework to apply architecture-specific tuning. For a range of stencil computations, we show that it achieves high performance for both the Intel Knights Landing (Xeon Phi) CPU, and the NVIDIA GPUs [78, 79]. This year we extended the library in multiple ways. We show that the indirection in the brick data layout permits distinct physical and logical data layouts; we can therefore store the bricks in memory to reduce the data movement of packing and unpacking during cross-node communication. We have demonstrated strong scaling by reducing communication time.

Next Steps We will continue to work with ECP application teams to integrate our tools with their efforts. In particular, we are integrating bricks into the Proto system, used in subsurface flows. Moreover, we are developing online tuning capability for ytopt/SuRF.

4.2.16 WBS 2.3.2.10 PROTEAS-TUNE - Bricks

Overview We have developed a source-to-source structured grid data framework, Bricks, to address the growing gap between memory and computational performance on pre-exascale systems. The approach uses standard C++ code to express stencil loops, for example, then transforms the code to use a different memory alignment and ghost zone region to optimize memory and communication performance specifically for that kernel [78, 79, 80]. This also provides an opportunity to inject architecture-specific code transformations, that can take advantage of SIMD/SIMT instructions, threading, virtual memory layouts, and specific parameters that are needed for each platform’s optimal performance. This is a powerful paradigm for having both a correct legal code base using standard tools and, in combination with the autotuning tools previously described, the ability to achieve portable performance on many different platforms with auto-generated code transformations.

Key Challenges Bricks require three primary ingredients for performance portability:

- (1) Stencil kernel metadata that includes stencil radius, dimensionality, neighbor dependences, and other memory access patterns is needed. For example, for communication, the optimal layout depends on the extent of stencil corner coupling and symmetry or reuse. For other kernels, it can involve making sure certain directions are contiguous, as in 1D FFTs or operators on neighboring vector components.
- (2) A transformation profitability model based on the architecture characteristics and benchmarks is needed to determine what transformations could improve overall throughput—not just maximize flops or bytes moved. This can also be explored using roofline models, auto-tuning, communication-avoiding techniques, etc.
- (3) Back-end optimizations and benchmarks are needed to understand which architecture-specific transformations achieve the best roofline performance and how to isolate and compare those with a known benchmark problem. For example, if there are special hardware capabilities, like vector shuffle, or OS support for memory mmap or prefetch, that are required to obtain peak performance.

Solution Strategy With Bricks, we have developed a data layout library, code generator, and communication primitives for both stencil computations and ghost zone communication. Recent trends in computer architecture that favor computation over data movement incentivize high-order methods and lower-precision data. Paradoxically, high-order codes can be challenging for compilers/optimization to attain high performance due to large amounts of intermediate data that exceeds available temporary storage. Bricks enable high performance and make fine-grained data reuse and memory access information known at compile or run-time. Integration with autotuning attains performance that is close to Roofline performance bound for both manycore CPU and GPU architectures.

In addition, we have used our Bricks source-to-source transformation technique to eliminate the cost of MPI packing/unpacking on CPUs and GPUs, which improves strong scaling for block semi-structured applications, is performance-portable, and is directly relevant to applications with many DoF’s per grid point (such as in combustion and multi-physics codes). This is because for most stencil codes, strong scaling is limited by the communication of ghost zone values or exchanged between processors and nodes, which is required for iterative algorithms or time integrators. Typically, this means each MPI rank requires packing before sending—copying a subset of local arrays into an MPI message buffer—and then unpacking (copying buffers to array subset) after receiving data. These operations on the ghost zone skin can introduce significant

latency and is a blocking operation that, in the limit of strong scaling, cannot be hidden by overlapping communication and computation. We have eliminated un/packing, and also introduced a novel technique on CPU to significantly reduce the number of messages, using an indirect mapping of memory to MPI buffers (Figure 43).

Recent Progress We have recently extended Bricks to incorporate run-time information on stencils, and use JIT (just-in-time) compilation to dynamically generate, compile, and link optimized kernels. We have also extended the approach to 6D computations with mixed FFT and stencil calculations, such as are seen in Fusion science. This required us to evaluate data types (real, complex) and precision (double, single) in the context of different architectures' performance-optimal parameters such as vector lengths and page/cache size. We have extended SIMD code generation to achieve performance portability for high-order stencils for both CPUs with wide SIMD units (Intel Knights Landing and later generations, ARM SVE) and GPUs (NVIDIA Ampere, AMD MI100, and Intel Gen11). We have demonstrated compatibility with standard C++ and most major GPU programming models, including CUDA, HIP, DPC++/SYCL, and OpenCL.

Next Steps For FY22, we are extending Bricks to other application patterns, including block-structured AMR, multigrid solvers, and systems of sparse and block-sparse (non-)linear systems. For these, the primary focus will be on investigating the profitability models, code transformations, and auto-tuning the kernels. As more diverse architectures and benchmarks become available within the ECP program (AMD GPU, Intel GPU, NVIDIA Ampere), we will develop transformations that provide better performance portability. We will be building up to portable application performance and load balancing; this is a complex trade-off between all kernels in a given code, and will be very application- and architecture-dependent.

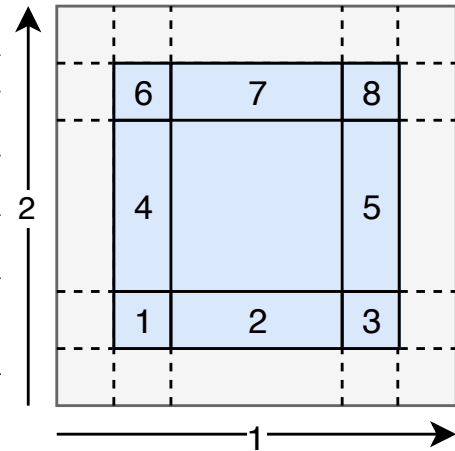


Figure 43: Bricks can be used to map memory onto regions that eliminate ghost zone packing and MPI message count (2D example).

4.2.17 WBS 2.3.2.10 PROTEAS-TUNE - TAU Performance System

Overview The TAU Performance System is a versatile profiling and tracing toolkit that supports performance instrumentation, measurement, and analysis. It is a robust, portable, and scalable performance tool for use in parallel programs and systems over several technology generations. It is a ubiquitous performance tool suite for shared-memory and message-passing parallel applications written in C++, C, Fortran, Java, Python, UPC, and Chapel. In the PROTEAS project, TAU is being extended to support compiler-based instrumentation for the LLVM C, C++, and Fortran compilers using higher-level intermediate language representation. TAU is also targeting support for performance evaluation of directive based compilation solutions using OpenARC and it will support comprehensive performance evaluation of NVM based HPC systems. Through these and other efforts, our objective to better support parallel runtime systems such as OpenMP, OpenACC, Kokkos, ROCm, and CUDA in TAU. Figures 44 and 45 give examples of using TAU's parallel profile analysis tool, ParaProf.

Key Challenges Scalable Heterogeneous Computing (SHC) platforms are gaining popularity, but it is becoming more and more complex to program these systems effectively and to evaluate their performance at scale. Performance engineering of applications must take into account multi-layered language and runtime systems, while mapping low-level actions to high-level programming abstractions. Runtime systems such as Kokkos can shield the complexities of programming SHC systems from the programmers, but pose challenges to performance evaluation tools. Better integration of performance technology is required. Exposing parallelism to compilers using higher level constructs in the intermediate language provides additional opportunities for instrumentation and mapping of performance data. It also makes possible developing new capabilities for observing multiple layers of memory hierarchy and I/O subsystems, especially for NVM-based HPC systems.

Solution Strategy Compilers and runtime systems can expose several opportunities for performance instrumentation tools such as TAU. For instance, using the OpenACC profiling interface, TAU can tap into a wealth of information during kernel execution on accelerators as well measure data transfers between the host and devices. This can highlight when and where these data transfers occur and how long they last. By implementing compiler-based instrumentation of LLVM compilers with TAU, it is possible to how the precise exclusive and inclusive duration of routines for programs written in C, C++, and Fortran. Furthermore, we can take advantage of the Kokkos profiling interface to help map lower level performance data to higher level Kokkos constructs that are relevant to programmers. The instrumentation at the runtime system level can be achieved by transparently injecting the TAU Dynamic Shared Object (DSO) in the address space of the executing application. This requires no modification to the application source code or the executable.

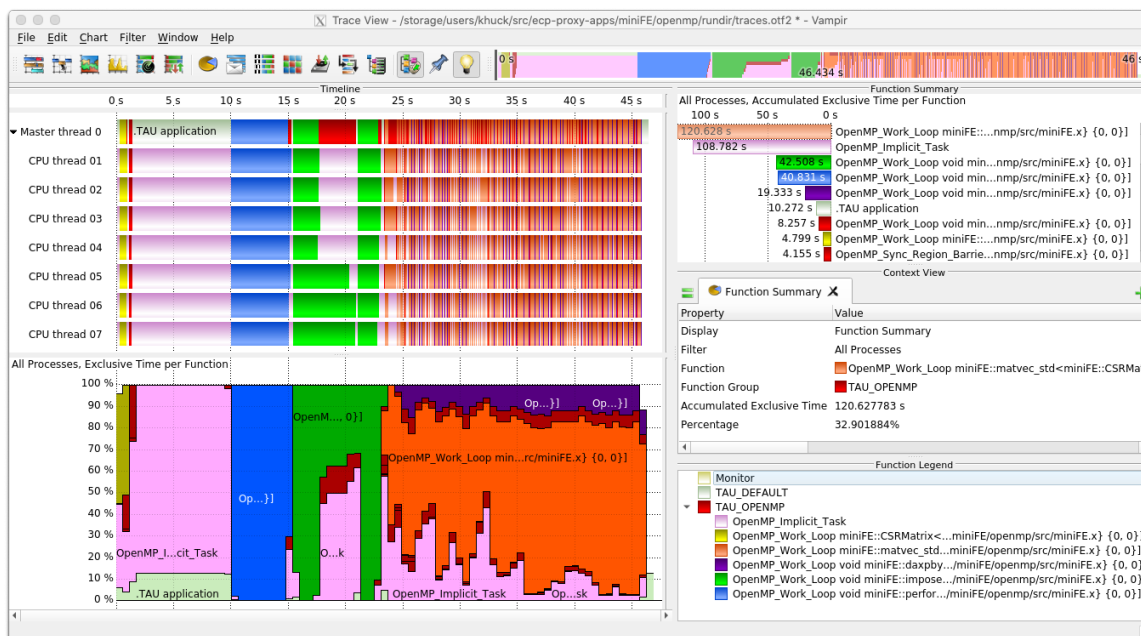


Figure 44: TAU was used to collect profiles and traces of ECP proxy applications like miniFE (trace shown in Vampir), observing OpenMP parallel regions, loops and synchronization without application instrumentation.

Recent Progress Recent progress in this area includes several key developments in support for CUDA, OpenMP, Clacc, and HIP, and PDT has been ported to pre-exascale systems. More details are provided below.

- Updated support in TAU for NVIDIA A100 GPUs with support for CUDA 11+.
- Updated OMPT support to OpenMP 5.0, tested with ECP Proxy applications miniFE and miniQMC as shown in the Vampir [81] trace viewer in Figure 44. Tentative OpenMP 5.1 support prototyped when runtimes provide the tool callbacks.
- Updated profiling support for OpenACC events provided by the Clacc compiler as shown in the Vampir trace viewer in Figure 45.
- Updated support for AMD GPUs with ROCm 3.3 - 4.2, tested on Spock (ORNL).
- PDT has been ported to pre-exascale systems. PDT is typically installed as the static analysis component of the TAU Performance System and is available on JLSE, ALCF, OLCF, NERSC, LLNL, LANL and Sandia systems and is installed as part of the system software stack.

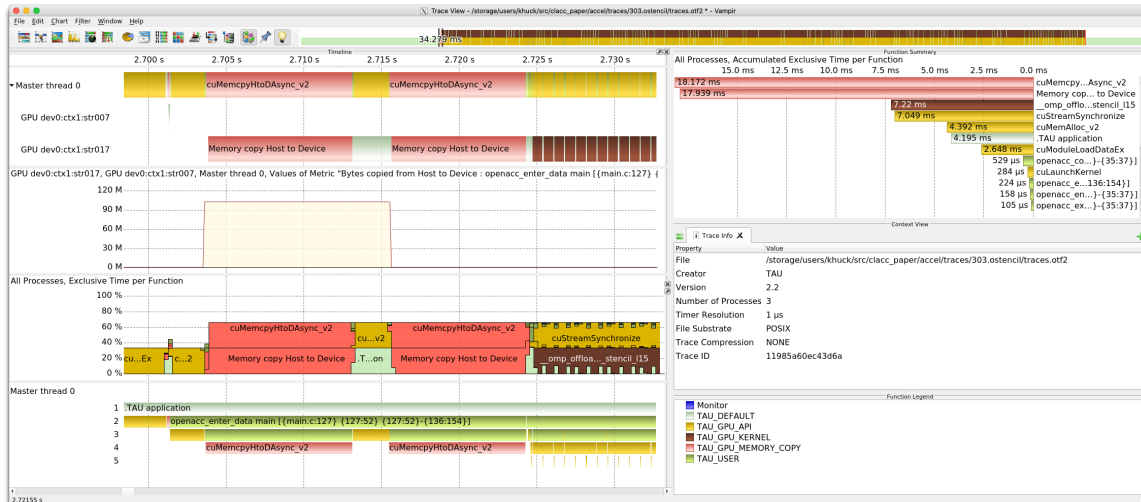


Figure 45: TAU was used to collect profiles and traces of OpenACC benchmarks (303.stencil trace shown in Vampir), observing OpenACC regions and device offload events without application instrumentation.

- Implemented and tested support for Intel OneAPI with Level Zero and the HPE Cray platform with AMD GPUs.
- Updated TAU in E4S to support NVIDIA, AMD, and Intel GPUs. Both Docker and Singularity images posted on E4S.io website include TAU with support for NVIDIA, AMD and Intel GPUs.
- Implemented an LLVM module for selective instrumentation of C/C++ using TAU, tested with LLVM versions 6 through 12 and Clacc.
- Extended OpenACC and OpenMP interoperable framework, (publication presented at SC20).
- New TAU source-to-source auto-instrumentation support in TAU for C++ by replacing current parser front-end with LLVM based solution.
- Added Fortran support for both PDT-based source instrumentation and compiler-based instrumentation with an LLVM selective instrumentation module.

Next Steps The following efforts have been identified for the next phase of the project.

- Implement new Profiling API and Perfworks Metrics API for CUDA/CUPTI 11+.
- Test and update prototype measurement for OpenMP 5.1 regions executed on target devices (when support is available from runtimes).
- Update and debug support for Intel OneAPI with Level Zero and the HPE Cray platform with AMD GPUs.
- Continued outreach activities to demonstrate comprehensive performance evaluation support in TAU for OpenARC, OpenACC, LLVM compiler-based instrumentation, CUDA, Kokkos, ROCm, and NVM based programming frameworks for SHC platforms.
- Continued integration of TAU and PROTEAS-TUNE projects in the E4S.
- Integrate new TAU source-to-source auto-instrumentation support in TAU for C++ by replacing current parser front-end with LLVM based solution for systems where Clang compiler support available.

4.2.18 WBS 2.3.2.10 PROTEAS-TUNE - PAPYRUS: Parallel Aggregate Persistent Storage

Overview Papyrus is a programming system that provides features for scalable, aggregate, persistent memory in an extreme-scale system for typical HPC usage scenarios. Papyrus provides a portable and scalable programming interface to access and manage parallel data structures on the distributed NVM storage. Papyrus allows the programmers to exploit large aggregate NVM space in the system without handling complex communication, synchronization, replication, and consistency models. Papyrus consists of three components, virtual file system (VFS) [82], C++ template container library (TCL) [82], and key-value store (KV) [83]. (1) PapyrusVFS provides a uniform aggregate NVM storage image for the different types of NVM architectures. It presents an illusion of a single large NVM storage for all NVM devices available in the distributed system. Unlike other traditional kernel-level VFSs, PapyrusVFS is a lightweight user-level VFS, which is provided as a library so that applications can link to or dynamically load it. PapyrusVFS implements a subset of POSIX API related to file I/O. (2) PapyrusTCL provides a high-level container programming interface whose data elements can be distributed to multiple NVM nodes. PapyrusTCL provides three containers, including map, vector, and matrix, implemented as C++ templates. PapyrusTCL is built on top of PapyrusVFS. This enables PapyrusTCL to be decoupled from a specific NVM architecture and to present a high-level programming interface whose data elements are distributed across multiple NVM nodes transparently. (3) PapyrusKV is a novel embedded KVS implemented specifically for HPC architectures and applications to provide scalability, replication, consistency, and high performance, and so that they can be customized by the application. It stores keys and values in arbitrary byte arrays across multiple NVMs. PapyrusKV provides configurable consistency technique controlled by the application during the program execution dynamically to meet application-specific requirements and/or needs. It also supports fault tolerance and streamlined workflow by leveraging NVM's persistence property.

Key Challenges In HPC, NVM is quickly becoming a necessary component of future systems, driven, in part, by the projections of very limited DRAM main memory per node and plateauing I/O bandwidth. More concretely, recent DOE systems, such as NERSC's Cori, LANL/Sandia's Trinity, LLNL's Sierra, OLCF's Summit, TACC's Stampede2, and ALCF's Theta, include some form of NVM. This NVM will be used in two fundamental ways. First, it will be used as a cache for I/O to and from the traditional HDD-based external parallel file systems. In this case, most scientists believe that the caching can be implemented transparently, shielding complexity from the applications and users. Second, NVM will be used as an extended memory to provide applications with access to vast amounts of memory capacity beyond what is feasible with DRAM main memory. More interestingly, in HPC, this extended memory can be aggregated into a much larger, scalable memory space than that provided by a single node alone. In this second case, however, no portable and scalable programming systems exist.

Solution Strategy Our key goals for Papyrus are high performance, scalability, portability, interoperability with existing programming models, and application customizability. First, **high performance** is a clear need in HPC. The design of Papyrus should provide the opportunity to exploit NVM resources efficiently. Second, **scalability** is important in HPC as most of the applications must run on large sectors of the systems - thousands to hundreds of thousands of processors. Papyrus should not inhibit scalability; it should provide an interface that is able to scale as the application and system do. Third, **portability** is a necessary requirement because HPC applications must be able to run on multiple, diverse platforms at any given time. The upcoming DOE systems all have NVM integrated into the systems in different ways. Papyrus must provide both functional portability and performance portability across systems with different architectures. Fourth, **interoperability** is a practical requirement of HPC applications. Papyrus must be designed so that it can be incrementally introduced into an application without conflicting with existing HPC programming models and languages like MPI, UPC, OpenMP, OpenACC, C, C++, and Fortran. Furthermore, Papyrus should leverage characteristics of these other programming models when possible. Interoperability allows programmers to adopt Papyrus incrementally in legacy MPI applications avoiding major rewrites of the application. Fifth, **application customizability** is a key requirement to achieve high performance and scalability. HPC applications have many different usage scenarios, and thus Papyrus should have customizable parameters for key features that impact other important properties like performance and scalability.

Recent Progress Recent progress in this area includes several key developments in data compression, encryption, tweaks to Papyrus, and performance evaluation.

- Added data compression and encryption to Papyrus [84]. Our compression technique exploits deep memory hierarchy in an HPC system to achieve both storage reduction and performance improvement. Our encryption technique provides a practical level of security and enables sharing of sensitive data securely in complex scientific workflows with nearly imperceptible cost.
- Redesigned and optimized Papyrus to support multidimensional tables.
- Performed preliminary evaluation on OLCF’s Summit supercomputer.

Next Steps The following efforts have been identified for the next phase of the project.

- Versioning can be used to provide new levels of reliability and performance optimization. We will design and implement versioning in Papyrus.
- New APIs and hardware support is being developed for NVM technologies; we are implementing optimizations in Papyrus to take advantage of these advances.

4.2.19 WBS 2.3.2.10 PROTEAS-TUNE - SYCL

Overview OpenCL is an open standard maintained by the Khronos group. It offers programming portability across a wide range of software and hardware for graphics processing units (GPUs), multi-core processors (CPUs), and other accelerators. As opposed to the OpenCL programming model in which host and device codes are written in two languages, the SYCL standard specifies a cross-platform abstraction layer that enables programming of heterogeneous computing system using standard C++. It can combine host and device codes for an application in a type-safe way to improve development productivity. SYCL is a promising programming model for exascale computing. The relevant topics are migration from CUDA only to SYCL, performance and expressiveness of SYCL programs, maturity of SYCL compilers, and accesses to SYCL programs.

The PROTEAS-TUNE objectives for SYCL are as follows:

1. Develop a SYCL suite comprised of kernels from open-source benchmarks, scientific, and machine learning applications
2. Evaluate the performance of these kernels with contemporary SYCL implementations on heterogeneous computing platforms
3. Understand the impacts of SYCL compilers and computing platforms upon the performance gaps of these kernels
4. Propose SYCL features that can improve functional and performance portability
5. Engage with vendors, facilities, universities, and communities for the development of SYCL applications and compilers

Key Challenges Acknowledging CUDA’s established presence in high-performance computing, SYCL has been striving for a portability-enhancing path for a wider set of platforms. While SYCL can achieve functional portability, it does not solve performance portability. To address the challenge, understanding the applications, programming models, SYCL features, SYCL compilers, and the architectures of heterogeneous computing platforms are critical. There are many scientific and AI applications. Major programming models for the target platforms are CUDA, HIP, OpenMP, and SYCL. Major SYCL features are extension to OpenCL C, single source, USM and buffer styles, and asynchronous programming. Major SYCL compilers are DPC++ with OpenCL and Level Zero backends, DPC++ with CUDA and HIP support, hipSYCL, and ComputeCpp. AMD, Intel, and Nvidia GPUs are different in their computing architectures. CPUs, GPUs, and FPGAs have fundamentally different architectures. The combination of applications, languages, features, toolchains, and architectures will characterize the performance of a SYCL program.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each effort.

- Develop a diverse set of SYCL programs with compute- and memory-bound kernels for performance analysis
- Have a good understanding of the characteristics of these programs
- Evaluate the performance of SYCL programs with the latest SYCL compilers on computing platforms
- Analyze the performance of SYCL kernels through performance profilers
- Identify and summarize these performance differences
- Engage with SYCL compiler developers to fix bugs and improve kernel performance
- Collaborate with PROTEAS teams and SYCL developers on performance optimization and tuning

Recent Progress Recent progress in this area includes several key developments listed below.

- Investigated the use of vendor and academic conversion tools by evaluating CUDA portability with HIPCL and DPCT [85]
- Investigated the performance of integer sum reduction in SYCL and CUDA on GPUs [86]
- Developing 200+ SYCL programs in the open-source GitHub repository (<https://github.com/zjin-lcf/HeCBench.git>)

Next Steps The following efforts have been identified for the next phase of the project.

- Continue developing SYCL programs with codes of interest to ECP
- Evaluate kernel performance with SYCL compilers on target platforms
- Analyze the performance of SYCL kernels through performance profilers
- Sum up the optimization techniques for SYCL kernels on target platforms
- Engage with SYCL developers to fix bugs and improve performance and portability
- Support compiler installation, feature requests, and bug reporting for ECP users

4.2.20 WBS 2.3.2.11 SOLLVE

Overview OpenMP is a directive-based API for intranode programming that is widely used to parallelize and accelerate large complex scientific applications including the ones that are of importance to ECP and beyond. Implementations of OpenMP and tools to facilitate OpenMP application development are available in all DOE LCFs. The specification is supported by a number of community of vendors, research labs, and academics who participate in the efforts of the OpenMP ARB and its Language Committee to evolve its features. The mission of the SOLLVE project is to further enhance OpenMP and its compiler and runtime implementations to meet the performance and productivity goals of ECP applications while these applications are being prepared to run on existing and upcoming large scale computing systems.

SOLLVE has been identifying OpenMP features and creating implementations to standardize them via active participation in the deliberations of the Language Committee. It is constructing a high-quality, robust OpenMP implementation based on the LLVM compiler. SOLLVE members have been working closely with the application developers in order to understand the needs the gaps in the specification, create use cases to motivate new feature requirements that would enable better parallelization and acceleration along with porting and migration of ECP applications to computing systems. SOLLVE has also been developing a verification and validation (V&V) suite to assess implementations and enable evaluations by DOE facilities including computing systems from ORNL, Argonne, NERSC as well pre-exascale testbeds, NSF infrastructure like Jeststeam and other academic systems.

SOLLVE plays a critical role in specifying, implementing, promoting, and deploying functionality that will enable ECP application developers to reach their goals using OpenMP.

Key Challenges Gaps in OpenMP functionality exist as a result of the rapid evolution of node architectures and base programming languages, as well as a lack of focus on performance portability before version 5.0. Since vendor representatives dominate the OpenMP Language Committee, effort is needed to secure their support with regard to the scope of the API, as well as the syntax and semantics of new features.

The API has greatly expanded in recent years as some of these gaps are closed, placing a large burden on its developers. The timely provision of robust implementations of new features that are critical for ECP is therefore particularly challenging. For performance portability, consistent approaches in multiple implementations is highly desirable. Interoperability concerns have emerged as a new challenge.

Given the lack of availability of implementations with features that target accelerators, many existing codes have used alternative APIs for GPUs: a significant effort will be required to replace those approaches by OpenMP. A broad effort is required to develop and apply best practices for new features and platforms.

Solution Strategy We address these challenges by focusing on the following primary activities.

Application Requirements Ongoing in-depth interactions with selected ECP application teams have resulted in a list of required extensions, some of which have been met by the recent 5.x specification. New needs are being identified, including performance of OpenMP features in applications. This work informs all other project activities by producing use cases, detailed feedback and example codes.

OpenMP Specification Evolution Members of the SOLLVE project are active participants in the OpenMP Language committee. The project creates early prototypes for new features based on ECP use cases, develops concrete proposals and submits them for standardization. Work has also been done to contribute to the OpenMP Examples document in the OpenMP Specification.

LLVM Compiler SOLLVE implements new OpenMP features in the LLVM compiler and develops analyses and transformations that enhance, and provide consistency to, OpenMP performance. Its open source solutions may be leveraged in vendor compilers. The compiler is available on LCF platforms.

OpenMP Runtime Systems BOLT, an experimental runtime built upon ultra-lightweight threading, addresses the need for efficient nested parallelism and improved task scheduling, and it develops better support for interoperability with MPI. BOLT is integrated and delivered with the project's LLVM compiler. A prototype runtime library for OpenMP loop scheduling across heterogeneous processors on a node is also being investigated.

Validation and Verification (V&V) A V&V suite is being developed that allows vendors, users and facilities to assess the coverage and standard compliance of OpenMP implementations. A ticket system for bug reporting and inquiries has also been deployed to facilitate interaction with end users.

Training and Outreach Tutorials and webinars are delivered to provide information on OpenMP features and their usage, as well as updating on the status of implementations. Deeper interaction with application programmers via hackathons supports the development of ECP codes using all available OpenMP features.

Recent Progress Figure 46 shows the latest progress on the 5 core SOLLVE thrust areas. The training and outreach activity is a cross-cutting effort which is supported by resources from SOLLVE and ECP Broader Engagement, with contributions by external collaborators, notably Lawrence Berkeley National Laboratory. A number of articles have also been published as part of the SOLLVE effort [87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 100, 105, 71].

Next Steps The following efforts have been identified for the next phase of the project.

Applications Continue to interact with ECP applications teams, evaluate implementations of new features and explore new requirements; identify best practices for the use of OpenMP on accelerators;

OpenMP Specification Continue work toward the next version of the standard via ECP-motivated feature development and participation in the OpenMP Language Committee: version 5.2 is underway and due for release November 2021;

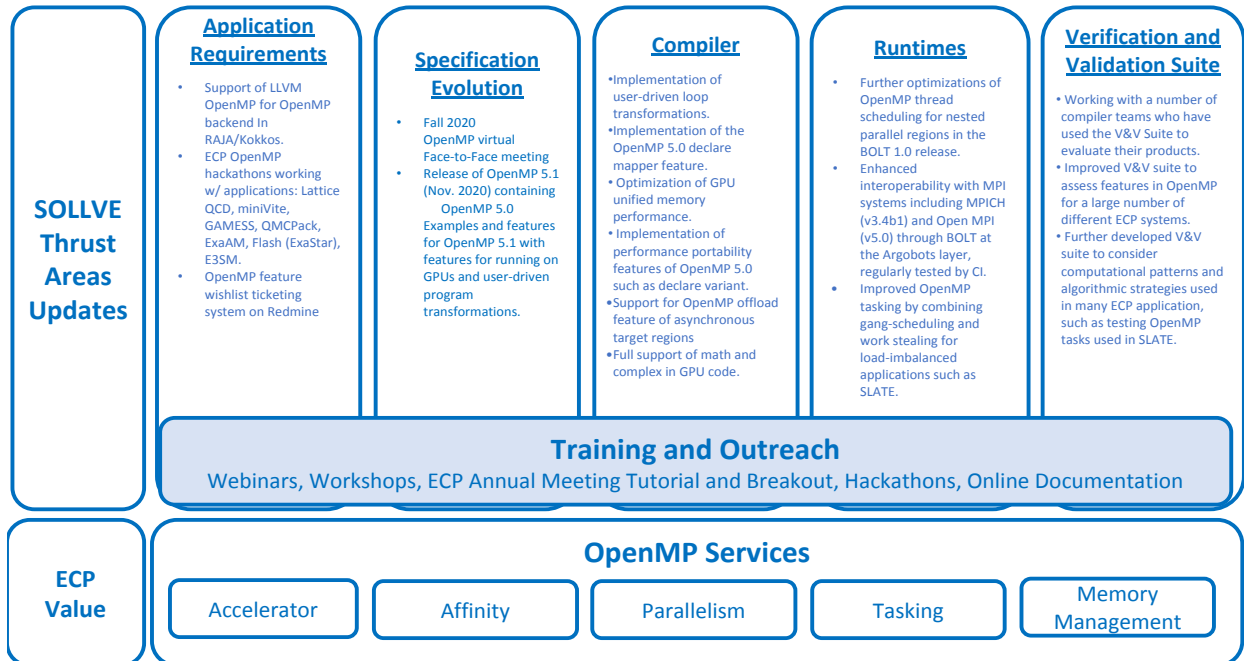


Figure 46: SOLLVE thrust area updates.

LLVM Compiler Improve performance of device offloading and optimize generation of code within target devices; generalize to enable reuse across multiple offloading architectures; develop infrastructure to support integration of Fortran front end; increase parallel region performance; implementation of OpenMP 5.1 loop transformations;

OpenMP Runtime Provide support for 5.x spec; address broader set of interoperability challenges including MPI+OpenMP codes using BOLT; address advanced tasking requirements, including on heterogeneous nodes of ECP Systems.

V&V Suite Continue expanding the coverage of the V&V Suite, with focus on 5.1 features along with additions and corrections to 4.5 and 5.1 tests as OpenMP implementations mature; expand Fortran tests; work with ARB Examples Committee; improve ALCF toolchains; more vendor interactions.

4.2.21 WBS 2.3.2.11 Argobots: Flexible, High-Performance Lightweight Threading

Note: This work is now dormant as the lead developers in Argonne National Laboratory have either left Argonne or been moved to other projects. The last status update is provided below for informational purposes.

Overview Efficiently supporting massive on-node parallelism demands highly flexible and lightweight threading and tasking runtimes. At the same time, existing lightweight abstractions have shortcomings while delivering generality and specialization. Our group at Argonne developed a lightweight, low-level threading and tasking framework, called Argobots. The key focus areas of this project are: (1) To provide a framework that offers powerful capabilities for users to allow efficient translation of high-level abstractions to low-level implementations. (2) To provide interoperability with other programming systems such as OpenMP and MPI as well as with other software components (e.g., I/O services). (3) To provide a programming framework that manages hardware resources more efficiently and reduce interference with co-located applications.

Key Challenges Several user-level threading and tasking models have been proposed in the past to address the shortcomings of OS-level threads, primarily with respect to cost and flexibility. Their lightweight nature

and flexible generic interface play an important role at managing efficiently the massive concurrency expected at the Exascale level. Existing user-level threading and tasking models, however, are either too specific to applications or architectures or are not powerful or flexible. Existing runtimes tailored for generic use [106, 107, 108, 109, 110, 111, 112, 113, 114] are suitable as common frameworks to facilitate portability and interoperability but offer insufficient flexibility to efficiently capture higher-level abstractions, while specialized runtimes [115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125] are tailored to specific environment.

Solution Strategy Argobots offers a carefully designed execution model that balances generality of functionality with providing a rich set of controls to allow specialization by end users or high-level programming models [126]. Delivering high performance in Argobots while providing a rich set of capabilities is achieved by heavily optimizing critical paths as well as by exposing configuration knobs and a rich API, which allow users to trim unnecessary costs. Furthermore, Argobots honors high degrees of expressibility in three key ways.

First, Argobots captures the requirements of different work units, which are the most basic manageable entities. Work units that require private stacks and context-saving capabilities, referred to as user-level threads (ULTs, also called coroutines or fibers), are fully fledged threads usable in any context. Tasklets do not require private stacks. They are more lightweight than ULTs because they do not incur context saving and stack management overheads. Tasklets, however, are restrictive; they can be executed only as atomic work units that run to completion without context switching.

Second, Argobots also exposes hardware computational units through execution streams (ESes) as OS-level threads to execute work units. Unlike existing generic runtimes, ESs are exposed to and manageable by users.

Finally, Argobots allows full control over work unit management. Users can freely manage scheduling and mapping of work units to ESs through thread pool management, and thus achieving the desired behavior. Figure 47 illustrates the various building blocks in the Argobots framework and the interactions between them to build a hypothetical system.

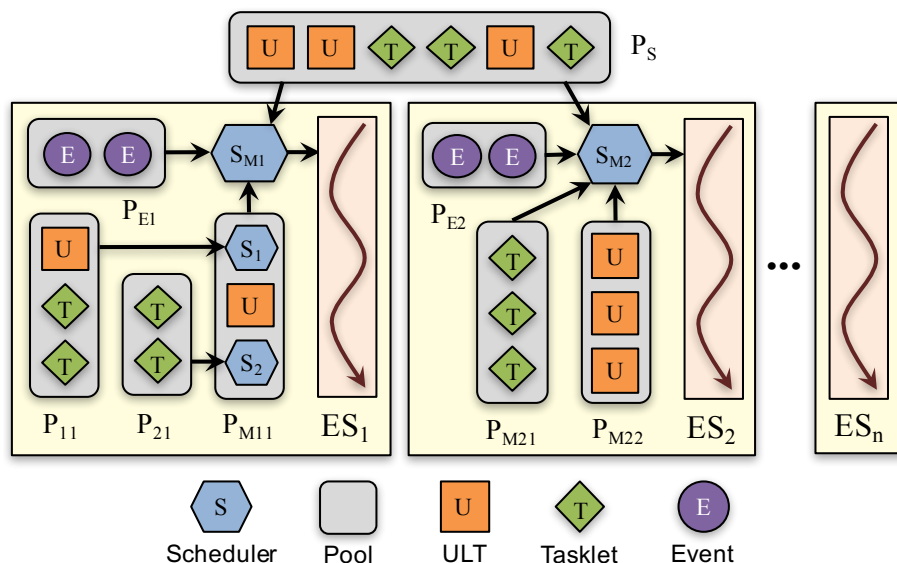


Figure 47: Argobots execution model.

Recent Progress Threading overheads are crucial for fine-grained parallel applications and runtimes running in massively parallel environments. We have found that the timing of yield operations highly affects the performance of lightweight threads [127], but other factors remained unexplored. We further optimized fork-join overheads by exploring new threading methods with respect to stack allocation timing and scheduling policies and a wider range of modern hardware architectures. Our evaluation shows that our child-first scheduling yields promising results for deep and narrow recursive task-parallel programs while the parent-first scheduling is good for flat parallelism. Our study helps users and application developers choose

the best threading methods that fit their hardware architectures and application workloads and maximize the scalability [128].

Integration with other runtime systems is fundamentally important for the Argobots project. BOLT, a SOLLVE OpenMP runtime over Argobots [40], is one of the most successful parallel programming systems using Argobots. Our enhancements of Argobots threads lowers the cost of OpenMP threading and tasking. The Argobots project continues to improve interoperability with communication layers such as MPI runtimes (e.g., MPICH and Open MPI) and Margo, a Mercury RPC over Argobots. To help their performance analysis, our latest Argobots 1.1 release includes a lightweight yet powerful profiling interface, which helps runtime developers pinpointing a performance problem in these systems. I/O service is one of the most important application areas for Argobots. Intel DAOS, a next- generation high-performance storage system developed by Intel, uses Argobots to efficiently handle asynchronous I/O messages. We are working together to improve Argobots by providing a better debugging interface such as stack dump features. Our CI testing has been extended for various CPU architectures, operating systems, and compilers to cover most of the DOE HPC platforms. Thanks to our CI, Argobots 1.1a1 works on major UNIX-based platforms including Ubuntu, FreeBSD, CentOS, macOS, and Solaris. Argobots supports most CPU architectures with special optimizations for Intel/AMD x86/64, ARMv8-A, and POWER 8 and 9. Argobots can be compiled with numerous C compilers including GCC, Clang, ICC (Intel), XLC (IBM), PGCC (PGI), Solaris Studio (Oracle), and Arm C Compiler for HPC (ARM).

Next Steps Argobots continues to implement new features and optimizations for application needs, while our substantial efforts will be made to promote integration and composition with other systems. Our major ongoing and planned steps are as follows.

1. Further integration with other applications and runtimes including MPI runtimes including MPICH and Open MPI. In collaboration with MPICH and Open MPI developers, we will further optimize their Argobots interoperability layers by utilizing user-level threading techniques.
2. Enhanced interoperability of multiple components that are not aware of Argobots. Unfortunately, not all applications are written for lightweight ULTs; some programs suffer from core starvation and, in the worst case, deadlocks if they are running on Argobots. To address this issue, we are investigating an approach that is as lightweight as the current Argobots ULTs while it has the OS-implicit preemption functionality that traditional OS-level threads have.

4.2.22 WBS 2.3.2.11 BOLT: Lightning Fast OpenMP

This work is now dormant as the lead developers in Argonne National Laboratory have either left Argonne or been moved to other projects. The last status update is provided below for informational purposes.

Overview OpenMP is central for several applications that target Exascale, including ECP applications, to exploit on-node computational resources. Unfortunately, current production OpenMP runtimes, such as those that ship with Intel and GNU compilers, are inadequate for the massive and fine-grained concurrency expected at the Exascale level. These runtimes rely on heavy-handed OS-level threading strategies that incur significant overheads at fine-grained levels and exacerbate interoperability issues between OpenMP and internode programming systems, such as MPI and OpenSHMEM. BOLT is a production quality OpenMP runtime (called BOLT) which has been developed within the SOLLVE project to address this issue by leveraging user-level threads instead of OS-level threads (e.g., Pthreads). Due to their lightweight nature, managing and scheduling user-level threads incurs significantly less overheads. Furthermore, interoperability between BOLT and internode programming systems opens up new optimization opportunities by promoting asynchrony and reducing hardware synchronization (atomics and memory barriers). Initial studies on this proposal can be found in [129, 130, 131]. This report briefly summarizes the issues in OpenMP runtimes that rely on OS-level threading, describes BOLT as the solution to this challenge, the current status in the BOLT effort, and the next steps for further improvements.

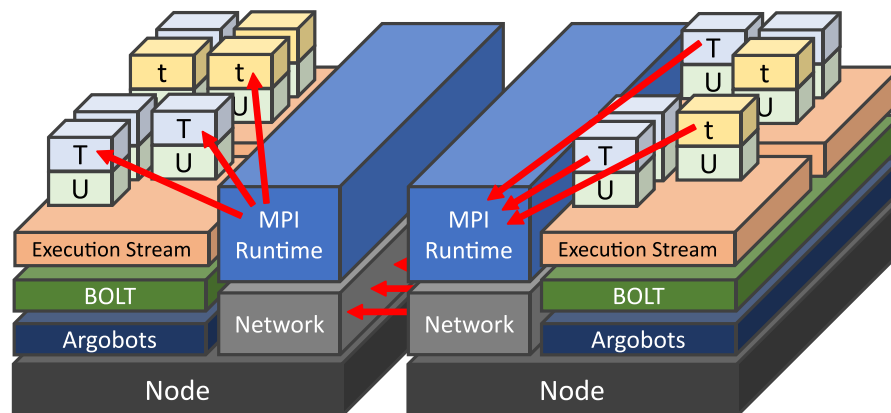


Figure 48: MPI+Threads interoperability of BOLT. OpenMP threads and tasks in BOLT interact MPI implementations via the Argobots layer.

Key Challenges The growing hardware concurrency in HPC cluster nodes is pushing applications to chunk work more fine-grained to expose parallelism opportunities. This is often achieved through nested parallelism either in the form of parallel regions or by explicit tasks. Nested parallel regions can potentially cause oversubscription of OS-level threads to CPUs and thus lead to expensive OS-level thread management. Such heavy costs usually outweigh the benefits of increased concurrency and thus compel the OpenMP programmer to avoid nested parallel regions altogether. Such workaround, however, not only causes poor resource utilization from insufficient parallelism but is also not always possible. For instance, the nested level could be outside the control of the user because it belongs to an external library that also uses OpenMP internally. Internode programming systems, such as MPI and OpenSHMEM, are not aware of OpenMP semantics, such as the notion of an OpenMP task. What these internode systems understand is the low-level threading layer used by OpenMP, such as Pthreads. This threading layer serves as the interoperability medium between OpenMP and the internode programming system and has a direct impact on performance. It is notoriously known that OS-level thread safety in production MPI libraries suffers significant performance issues. While continued progress on improving OS-level thread safety in these important internode programming systems is crucial for traditional interoperability, we propose in this work exploring an orthogonal direction that assumes a more lightweight interoperability layer.

Solution Strategy Both fine-grained parallelism and interoperability issues suffer from the heavy nature of working at the level of OS threads. Our solution to both challenges leverages user-level threads. Using user-level threads as the underlying threading layer for the OpenMP runtime offers a significantly better trade-off between high concurrency and thread management overheads. This allows users to generate fine-grained concurrency and oversubscription without worrying about the performance collapse that is observed in current OpenMP runtimes. Our OpenMP runtime, BOLT, is derived from the LLVM OpenMP runtime and leverages Argobots, a highly optimized lightweight threading library, as its underlying threading layer. OpenMP threads and tasks are spawned as Argobots work units and nested parallel regions are managed through an efficient work-stealing scheduler. Furthermore, new compiler hints and runtime optimizations have been developed to allow reducing thread management overheads even further [127, 128]. Interoperability improvements have also been demonstrated by having BOLT interoperate with an MPI libraries through the Argobots threading layer rather than OS-level threads. Results showed that this approach allows better communication progress and outperforms the traditional Pthreads-level interaction [126]. See Figure 48.

Recent Progress We improved the interoperability of BOLT with various MPI systems via lightweight threads, Argobots. Since most MPI runtimes including MPICH, Open MPI, and production MPI implementations that are derived from either MPICH or Open MPI assume OS-level threads as “Thread” in MPI+Thread, lightweight OpenMP runtimes based on lightweight threads failed to interoperate well with existing MPI

systems. To address this issue, we have implemented an abstracted threading layer for lightweight threads in these MPI runtimes so that users can choose OpenMP threads and tasks of BOLT as “Thread”.

Virtual communication interfaces (VCIs) in MPICH are mapped to logically parallel MPI operations, can benefit both multi-thread and multi-GPU configurations since both rely on VCIs for parallel communication. We developed VCI-aware OpenMP thread scheduling over BOLT via a user-level threading interoperability layer significantly reduces the locking overheads, realizing highly scalable MPI communication. The evaluation using ORNL Spock (Slingshot) shows significant performance improvement.

Our latest BOLT 1.0 release contains a large upgrade to be compatible with LLVM OpenMP 10.0, which further improves performance and functionalities especially for GPU offloading and new OpenMP 5.0 features. This release also contains scheduler improvement in BOLT and an upgrade of the Argobots package, which allows further lightweight fine-grained OpenMP threading and tasking. Interaction and integration are a critical piece for the BOLT project. We continue working on the SOLLVE Spack package so that this BOLT system is available on our target HPC systems and users can utilize BOLT for (1) ECP applications that have fine-grained parallelism such as nested parallel regions and tasking (e.g., ECP SLATE) and (2) runtime systems via the Argobots layer (such as MPICH and Open MPI we mentioned) that can take advantage of ULT’s lightweight synchronization for resource management.

Next Steps One of the largest advantages of BOLT is an underlying lightweight thread implementation, flexible scheduling, and high interoperability thanks to Argobots. The following list includes our next plans:

1. Explore opportunities for utilizing lightweight threads for other optimizations in the context of OpenMP. The main focus of BOLT has been the performance of fine-grained OpenMP threads, so we have not fully explored how BOLT could elevate the performance of other parallel units (e.g., data-dependent tasking and GPU offloading). We are planning to investigate room for optimizations and implement them with evaluation.
2. Investigate performance with large-scale applications. In order to find potential room for optimizations and evaluate the performance of BOLT in real large-scale workloads, we further investigate other SOLLVE components and ECP applications that can benefit from BOLT. Since most distributed systems rely on MPI for internode communication, we will also work on tighter integration with MPI to optimize the performance with MPI runtimes over BOLT.

4.2.23 WBS 2.3.2.11 LLVM Implementations

Overview The LLVM project is a collection of modular components for building compilers and toolchains. LLVM comes with support for OpenMP through the C/C++ frontend Clang. Execution is supported using OpenMP runtime libraries centered around `libomp` (for host execution) and `libomptarget` (for device execution). The key focus of this project is to implement OpenMP features in LLVM, enhance features, and provide best-in-class performance with the LLVM community version. As vendor compilers are built on top of LLVM, enhancements we integrate upstream are likely to be reused by one or multiple vendor compilers as well.

We will discuss asynchronous OpenMP offloading in detail now. Afterwards, we list other recent works and present results of enhanced OpenMP lowering for GPUs together with OpenMP-aware optimizations.

Key Challenges As most of the computational power is within GPUs, it is imperative for performance (per watt) to keep them occupied with productive work at all times. A meaningful approach is to perform as many computations as possible simultaneously. Even NVIDIA Fermi GPUs, which have been on the market for ten years, allow for concurrent execution of up to 16 GPU kernels on a single device. Asynchronous offloading is a promising technique to achieve such concurrency as it allows a single CPU thread to overlap memory movement, GPU computation, and the preparation of new GPU tasks on the CPU. Costly stalls between GPU computations (a.k.a. kernels) are avoided and the hardware can start the execution of an already prepared kernel as soon as the ones currently executed stop utilizing the entire device. OpenMP supports asynchronous offloading through the `nowait` clause on `target` directives, though compiler support still varies.

There are two existing designs that can do asynchronous offloading: regular task and detachable task. The regular task design is easy to implement, potentially even without compiler support, but it will fail to achieve the goal if there are no threads available to perform the offloading concurrently, which sometimes is unrealistic and restrictive. The detachable task design can be expected to provide consistently good results under most circumstances.

Solution Strategy We propose a new scheme to implement the `nowait` clause on `target` directives to achieve concurrent offloading. It is designed to provide good performance regardless of the context. Our approach utilizes otherwise hidden helper threads to provide consistent results across various use cases. In the hidden helper task design, a deferred target task is executed in its entirety by a thread that is not started by nor (in any language-defined way) visible to the user. These hidden helper threads form a team of threads that is implicitly created at program start and is only responsible for the execution of the special hidden helper tasks. We denote them in our pseudo code as `hht_task`. Such tasks are not too different from other deferred OpenMP tasks except that they are always executed by an implicit hidden helper thread. It is especially important that they participate in the dependence resolution like any other tasks generated by the encountering thread. Thus they are siblings to tasks generated by threads in the same team as the encountering thread. The hidden helper task concept is not tied to deferred target task but could help the definition or extension of the OpenMP specification. Figure 49 shows how the generic deferred target task is executed in this design.

```
#pragma omp hht_task shared(...) depend(...) ...
#pragma omp target map(...) ...
{ ... }
```

Figure 49: Conceptual lowering of the `target` directive to the hidden helper task design. A special `hht_task` is used and executed by an hidden helper thread while the offload part is made synchronous.

Our results show the hidden helper task design gains up to $6.7\times$ improvement on Summit supercomputer, and also provides comparable speedup to the commercial IBM XL C/C++ compiler. Most important, our design provides more functionalities and it can be used even if the native runtime has no asynchronous offloading capabilities.

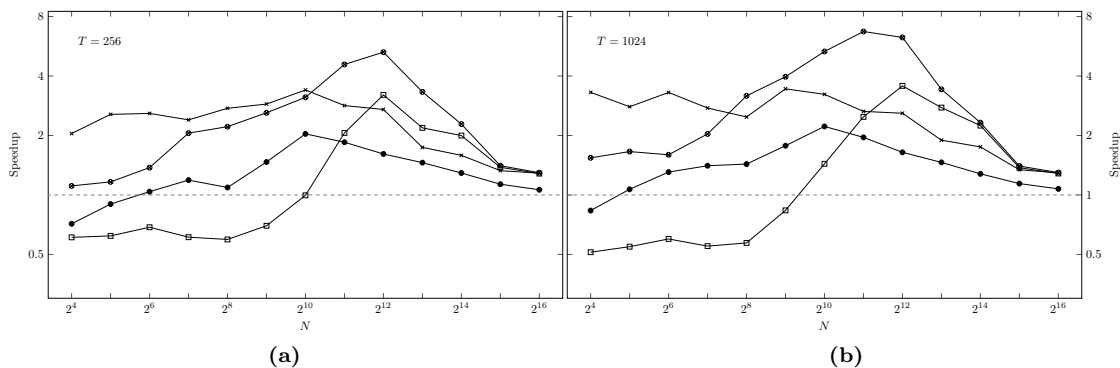


Figure 50: Speedup of concurrent execution with hidden helper tasks compared to vanilla LLVM with different benchmarks.

Recent Progress In recent work we implemented loop transformation constructions introduced in OpenMP 5.1 [70, 71], asynchronous offloading for OpenMP [132], efficient lowering of idiomatic OpenMP code to GPUs (under review), OpenMP-aware compiler optimizations with informative and actionable remarks for users (under review), a portable OpenMP device (`=gpu`) runtime written in OpenMP 5.1 (including atomic

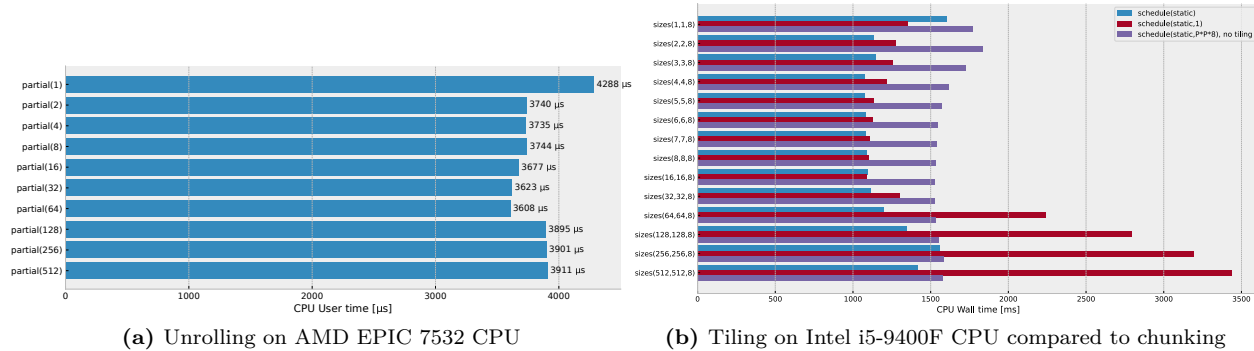


Figure 51: Performance impact of loop transformations on CPUs.

support) [133], a virtual GPU as debugging friendly offloading target on the host [134], improved diagnostics and execution information [135, 136].

We redone the OpenMP GPU code generation in LLVM/Clang [137] to improve performance and correctness. This work was complemented by a new LLVM/OpenMP GPU device runtime that helps us further close the performance gap compared to CUDA and other kernel languages [138]. Various efforts in improving development and debugging have also been integrated into LLVM/OpenMP [135, 134].

We implemented hidden helper task design in LLVM/OpenMP. Most of our work [133] (except device side dependence resolution) have been upstream and already part of LLVM since 12.0. Recently Intel also adopted our design and implementation in Intel’s latest OpenMP compiler [139].

The loop transformation constructs implemented in Clang include the unroll and tile directives. We demonstrate how these make speedup in the vicinity of $3\times$ (tiling on GPU) and an additional 20% become low-hanging fruits by just adding a directive [70] as illustrated in Figure 51. There are actually two implementations, one AST-based and a second using the OpenMPIRBuilder that can also be used other compilers such as Flang [71].

Testing of OpenMP offloading for NVIDIA and AMD GPUs has been added to LLVM’s Continuous Integration infrastructure. This will allow identifying changes that break OpenMP early during development, including those by non-OpenMP developers making maintenance easier.

GPU kernel times have been improved significantly since LLVM 12. Figure 52 illustrates the impact of different optimizations we integrated into LLVM and which are run by default for OpenMP codes. As shown, speedups of up to $13.35\times$ (RSBench, upper right) were reached while we get closer to CUDA performance across the board. More recent improvements caused benchmarks like XSBench (upper left) to also achieve CUDA performance when compiled with OpenMP offload via our development branch of LLVM/Clang.

LLVM/Clang 13.0.0 has been officially released on October 4, 2021, which includes most of the aforementioned improvements. The branch for LLVM/Clang 14.0.0 has been branched-off the development repository and is expected to be released March 15 2022.

Case study: LLVM/OpenMP performance for OpenMC We intensified our work with the OpenMC team on their portable OpenMP implementation of a Monte Carlo code. Most recent results obtained with LLVM/OpenMP show how our compile enhancements and application engagement improve performance manifold. Further, the early results on AMD GPUs are very promising and indicate that performance portability is achievable with LLVM/OpenMP. The performance evolution through our collaboration is summarized in Figure 53. JLSE testbeds from Argonne National Lab have been used for these results. A more detailed report was recently accepted to the PHYSOR conference.

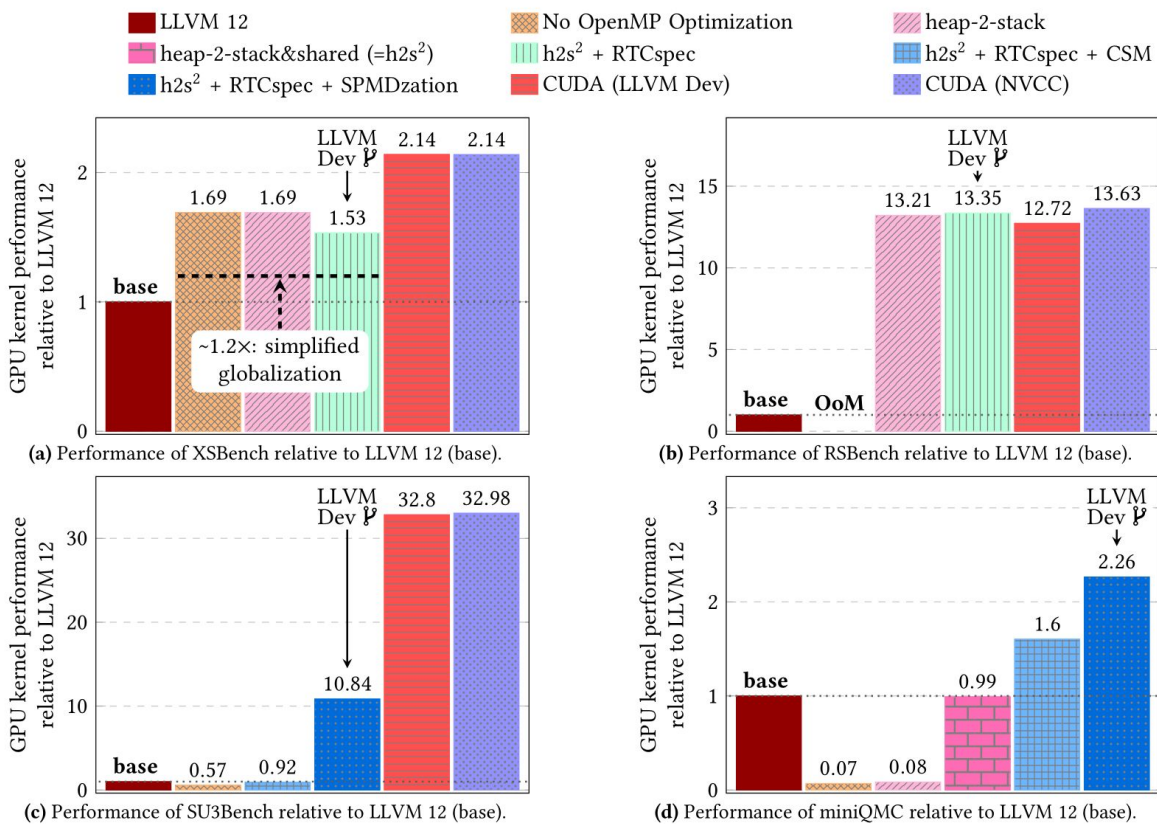


Figure 52: Kernel times for different ECP proxy apps when compiled with OpenMP offloading and CUDA. For the former differently optimized versions are shown and the results of a close to LLVM upstream branch is marked explicitly. All times are normalized against LLVM 12 OpenMP offloading performance.

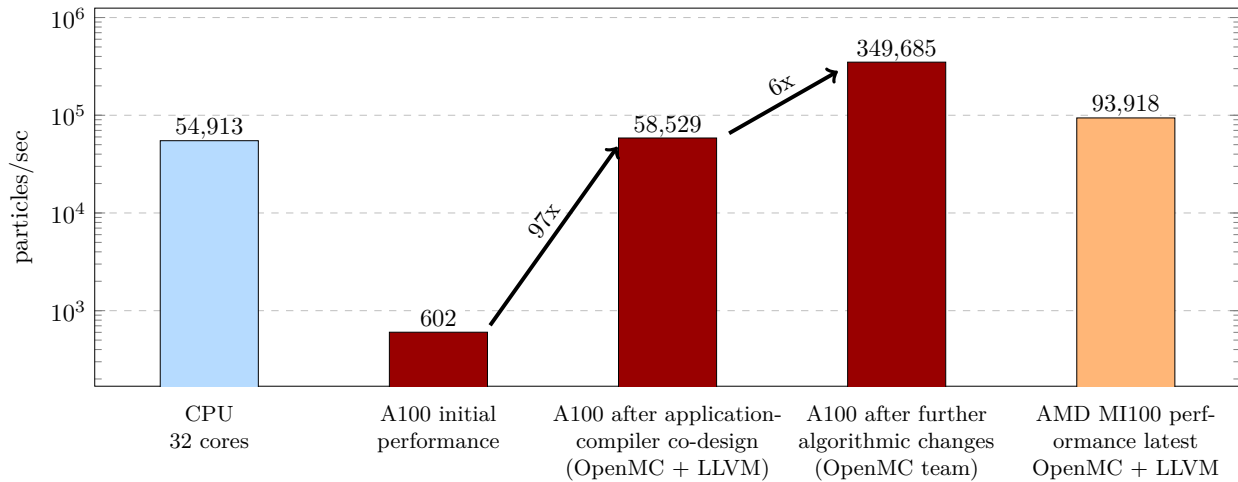


Figure 53: Performance results for the OpenMC [5] OpenMP offloading code that simulates transport of neutral particles using the Monte Carlo methodology. The application is part of the ExaSMR ECP project and known for its proxy applications (XSBench [6] and RSBench [7]). Results show the almost 100x speedup achieved by co-optimizing the full OpenMC application with the LLVM compiler for OpenMP offloading on NVIDIA A100 GPUs. In the process, changes to both OpenMC and LLVM were made, and the latter were often triggered via command line options or assumptions. Guided by compiler remarks, other applications could benefit similarly and with far less involvement from a compiler developer—but only if the remarks, and later the suggested code changes, are clear, actionable, and effective. For context, we also present the AMD MI100 numbers obtained with the latest versions of OpenMC and the LLVM/Clang compiler. Since the LLVM/OpenMP offloading backend for AMD GPU is new and only recently became stable, we expect substantial performance improvements as we start our tuning efforts. Even without, we believe the results shows how performance portability is certainly within reach for a full application using the LLVM/OpenMP offloading environment.

The data for the figure was generated and generously provided by John Tramm.

Next Steps We will keep working on supporting more new OpenMP 5.1 features in LLVM. To avoid redundant development of two OpenMP implementations, we will mature the development of the OpenMPIBuilder and make it the default OpenMP code generation for Clang as it is already for Flang.

Improvements to the OpenMP offloading ecosystem are planned, e.g., to extend the remote offloading capabilities we introduced and to further improve development and debugging of OpenMP GPU applications.

More OpenMP-aware optimizations are expected to materialize as we get the last OpenMP benchmarks on par with CUDA performance. We also plan to invest time in generic GPU optimizations soon.

We will implement proof-of-concept implementations of loop transformations that are planned for OpenMP 6.0, such as loop interchange, reversal, fusion, fission and index set splitting.

Experiences on Early Access Systems AutoDock-GPU is an application code for molecular docking often used to solve problems in the domains of biology and AI/ML on supercomputers including DoE's supercomputers, and it has recently (in 2020 and 2021) been used as the driver for simulations run on supercomputers for developing COVID-19 therapeutics [140]. AutoDock-GPU uses OpenMP offload features to run on a single GPU and has the ability for parallelize and schedule computation across multiple GPUs on a node through OpenMP. The OpenMP offload implementation of AutoDock-GPU was developed by members of the SOLLVE team. The application code has been experimented with using LLVM 12 OpenMP and LLVM 13 OpenMP on one node of the Spock Early Access System provided by ECP. Note that the LLVM 13 results were not in our SOLLVE FY21 Program Review slides and we only showed LLVM 12 then.

Figure 54a shows the wall clock timings of AutoDock-GPU when using 3 different sizes of ligands on one GPU of Spock, under LLVM OpenMP 12 and LLVM OpenMP 13. The results show that LLVM 13 OpenMP (rocm4.5) improves performance over LLVM 12 OpenMP (rocm4.2) of the AutoDock-GPU for the largest problem size, i.e., the 3er5 ligand, by 16.56%. The results show that the LLVM OpenMP implementation version generally has an impact to performance and suggests has a larger impact with larger problem size for this application.

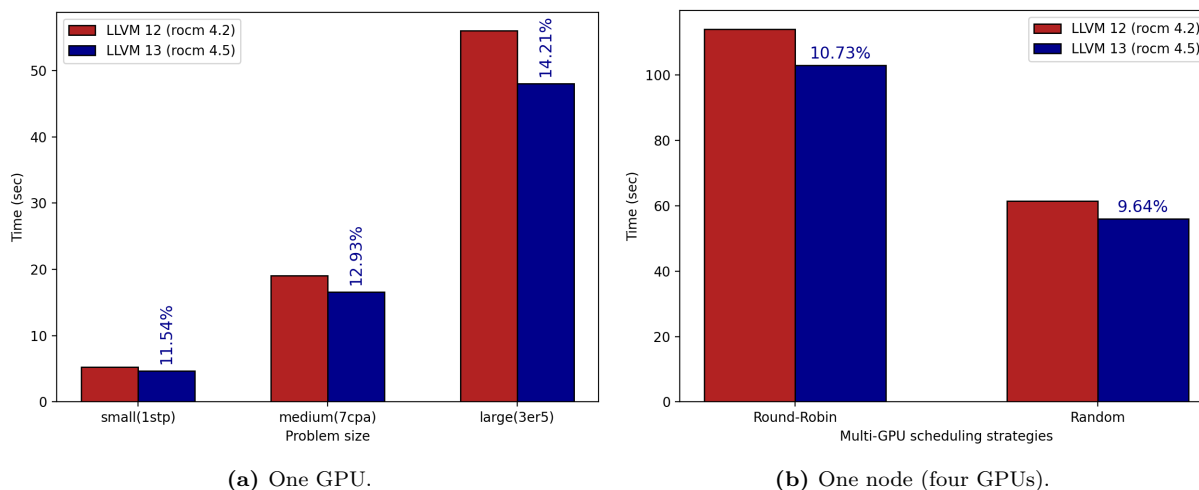


Figure 54: Running AutoDock-GPU with LLVM's OpenMP on Spock.

Figure 54b shows results for strategies for an OpenMP parallelization of AutoDock-GPU (using an assortment of the three ligands in Figure 54a) across the 4 GPUs of Spock. The figure specifically shows the difference in performance between a static assignment of the application's computation to GPUs and a dynamic assignment - or scheduling - of the application's computation across multiple GPUs of the node, as discussed in [87], and the impact of the LLVM OpenMP implementation version. The round-robin scheduler under LLVM 13 OpenMP (rocm4.5) improves performance over the round-robin scheduler in LLVM 12 OpenMP (rocm4.2) by 10.73%. Under LLVM 13's OpenMP (rocm4.5), using a dynamic assignment (labeled *random* in Figure 54b and where OpenMP chunks of a loop are assigned to randomly chosen GPU) as opposed to a static assignment (labeled *round-robin* in Figure 54b and where OpenMP chunks are assigned to a GPU

based on chunk number) cuts the execution time by 45.63%. By using the newer LLVM 13 over LLVM 12 and by using the random multiGPU scheduling strategy instead of the round-robin scheduling strategy, AutoDock-GPU is sped up by approximately $2.05\times$. Using the dynamic multiGPU scheduling strategy over the static one improves performance $4.5\times$ more than using the new LLVM OpenMP implementation version over the older one, but the newer LLVM OpenMP implementation version still helps improve performance substantially and by about 10%.

4.2.24 *WBS 2.3.2.11 Building tools to detect and debug bugs and performance issues with OMP offloading*

Overview Starting from the 4.0 version, OpenMP has introduced a new feature, target offloading, which enables programmers to leverage additional computing devices connected to the host (e.g., GPU, application-specific accelerator). With a group of hardware-agnostic constructs (device directives), target offloading makes it possible to write accelerated OpenMP applications in a performance-portable manner. However, concurrency bugs and performance issues may still arise due to incorrect usage of device directives. Our group at Georgia Tech has conducted a comprehensive study of such issues [141]. We found that most of these issues are related to the data mappings between the host and target device. For example, incorrect data mappings can lead to uninitialized memory accesses or even buffer overflow on the target device. In addition, programmers may also declare data mappings that incur unnecessary time and memory overhead due to redundant copies.

To help programmers detect and debug these concurrency bugs and performance issues, we have developed a toolset for OpenMP applications:

- OMPSan [142] is a static analyzer that can report all suspicious data mappings that may lead to concurrency bugs and memory issues;
- ARBALEST [143] is a dynamic analyzer that can pinpoint incorrect data mappings that result in concurrency bugs or memory issues at runtime;
- OmpMemOpt [144] is a static optimizer identifies redundant data mappings and replaces them by more optimized data mappings.

Key Challenges With respect to incorrect data mappings, multiple kinds of bugs may arise at runtime, such as use of uninitialized memory, use of stale data, buffer overflow, and data race. Currently, there exists no tool that can recognize all these bugs' behavior. Even if programmers apply multiple analysis tools when testing a single OpenMP application, some incorrect data mappings may still be missed since they may only trigger bugs in a specific thread interleaving. On the other hand, detecting redundant data mappings may also be challenging. OpenMP utilizes a reference-count algorithm to manage the lifecycle of each data mapping. Memory transfer is only carried out when a data mapping is first encountered. The analysis tool needs to correctly model this reference-count-based mechanism and all memory accesses related to data mappings to determine whether a data mapping is redundant.

Solution Strategy As a static analysis tool, OMPSan [142] compares the def-use information in an OpenMP application with the application's serial-elision version. OMPSan assumes that the serial-elision version contains the correct def-use information. Any inconsistent def-use chain incurred by data mappings can be statically reported as a potential bug.

ARBALEST is a dynamic analysis tool designed for incorrect data mappings. It is an extension to the Archer data race detector [145]. ARBALEST leverages a state-transition diagram to record the state of each data mapping, i.e., whether the memory section on the host/target device has a valid value. ARBALEST also utilizes the happens-before relations maintained by Archer to detect conflicting memory accesses when the runtime conducts memory transfer for a data mapping. According to the evaluation in [143], ARBALEST can accurately identify all incorrect data mappings with acceptable time and memory overhead.

OmpMemOpt[144] is a static optimizer that can generate optimal data mappings for an OpenMP application. The optimization framework in OmpMemOpt casts the problem of detection and removal of redundant data movements into a partial redundancy elimination (PRE) problem and applies the lazy

code motion technique to optimize these data movements. The evaluation on ten benchmarks shows that OmpMemOpt achieved a geometric speedup of 2.3x, and reduced on average 50% of the total bytes transferred between the host and GPU.

Recent Progress We have achieved a number of publications for the toolset. OMPSan has been presented at IWOMP 2019 and was selected as the best paper award recipient in that two workshops. The other two papers on ARBALEST and OmpMemOpt were presented at IPDPS 2021 and Euro-Par 2020 respectively.

Next Steps We are working with the LLVM OpenMP subcommunity to develop a new version of ARBALEST. The new implementation will be built on the latest LLVM and apply a number of optimization techniques to reduce the time overhead. Besides, we are also helping improve the OpenMP Tool Interface design based on our experience of developing analysis tools.

4.2.25 WBS 2.3.2.11 Validation and Verification Testsuite

Overview The OpenMP Validation and Verification Testsuite (OMPVV) presents the community with a test suite comprised of functional tests and application kernels that are written with the intention of verifying usability of OpenMP features. The test suite aims to accomplish several goals of equal importance. The first goal is to provide vendors with a straightforward way to examine the status of their implementations. The next goal is to support application developers in identifying whether their system can support the specific OpenMP features they wish to utilize. Both of these are accomplished by running the full test suite and generating a report summary, which provides the user with a simple pass-fail report for OpenMP features. These insights also include where the tests are failing (compile time or runtime) and what is the error encountered. Our full suite of OpenMP tests is made publicly available on Github [146] and full documentation is available on the corresponding website [147].

Key Challenges Every few years, the OpenMP Architecture Review Board releases new versions of their specification. These new releases of the specification, 5.0 in November of 2018, 5.1 in November of 2020, and 5.1 in November of 2021 introduce new clauses and directives that many application developers and general users are keen on utilizing. Thus, when the updates are published, we set out to create tests for each new feature or new directive. It takes time for the compiler developers to develop implementations for several of the new features, as a result there is a gap between versions released by the specification, implementations developed and implementations made available for the application developers. Due to this fact, we are often times writing test cases for new clauses and directives even before the implementations begin to exist. This can be quite a challenge since we cannot even compile or execute them right away, so would need to revisit the test cases as soon as the implementations are made available.

Solution Strategy For every new specification version we begin by classifying the priority of each new feature based on input from ECP and (sometimes) CAAR application teams. We also track LLVM's OpenMP implementation status and formulate a priority list for test implementation. As more features are implemented by LLVM, we continue to create functional tests cases and verify our own work by running the tests on several heterogeneous systems that represent multiple vendors and OpenMP compiler implementations. In previous years, we followed the same strategy for providing OpenMP 4.5 test coverage [90].

Recent Progress Over the past year, we have been able to provide tests (C, C++) that cover almost all of the new features introduced in OpenMP 5.0. We continue to improve our Fortran tests coverage for 5.0 features as well. We periodically report the pass-fail status and individual test results on state-of-the-art systems such as Oak Ridge National Lab's Summit and Spock computing systems as well as NERSC's Cori system [147]. This provides a useful tracking system for vendors and application developers alike. Also, we have created functional tests for several of the latest OpenMP 5.1 features, namely: atomic compare, C++ attribute specifier, declare variant, default first private, tiling etc. These have been implemented by LLVM and deemed of high importance to AD teams.

Next Steps We will continue C/C++ test development for OpenMP 5.1 specification while we maintain, improve and fix any issues discovered over our previously developed test. We have continued to follow the same strategy as we did for providing test coverage of 5.0 features. We periodically run our tests on the newest available compiler implementations that support OpenMP and maintain the results [147]. Additionally, we are tracking AD teams use of OpenMP features and incorporating application kernels in our tests that are outside the scope of individual feature tests but test a legal combination of OpenMP features that is critical for the application.

4.2.26 WBS 2.3.2.11 Training & Outreach in 2021

1. Annual ECP Meeting 2021 related

- “OpenMP Tutorial” by O. Hernandez, D. Oryspayev, T. Scogland, C. Bertoni, J. Doerfert, and V. Kale
- “ECP + SOLLVE COVID”
- “OpenMP Application Experiences BoF”
- “LLVM in ECP Short Stories About a Compiler Framework in HPC”
- “OpenMP Vendor BoF”
- “Session on Testing in ECP including OpenMP V&V Suite”
- “Tutorial: Autotuning PolyBench Benchmarks with LLVM Clang/Polly Loop Optimization Pragmas Using Bayesian Optimization”

2. “OpenMP Users Monthly Telecons⁶” led by D. Oryspayev

3. Weekly “OpenMP-in-LLVM” teleconference with DOE labs and vendors (AMD, Intel, Cray etc.)

4. Biweekly “OpenMP work in Flang” teleconference with DOE labs and vendors (AMD, ARM, NVIDIA etc.)

5. Hackathons

- “ECP SOLLVE + NERSC OpenMP Hackathon⁷”, January 22, and 27 – 29, 2021
 - Organizers: C. Daley, D. Oryspayev, K. Gott, H. He, O. Hernandez, and V. Kale
- “ECP OpenMP Virtual Hackathon⁸”, October 1, and 6 -8, 2021
 - Organizers: D. Oryspayev, J. Doerfert, S. Chandrasekaran, S. Pophale, T. Scogland, and V. Kale

6. ISC HPC’21⁹ related tutorials & presentations

- Tutorial: “OpenMP Common Core: Learning Parallelization of Real Applications from the Ground-Up” by M. Arenaz, B. Chapman, R. Budiardja, O. Hernandez, and D. Oryspayev.

7. OpenMP.org Webinars

- “A Compiler’s View of the OpenMP API¹⁰” by J. Doerfert.
- “The Leaders of OpenMP Discuss the Future of the OpenMP API¹¹” by M. Klemm and B. de Supinski.

⁶<https://www.openmp.org/events/ecp-sollve-openmp-monthly-teleconference/>

⁷<https://sites.google.com/view/ecpomphackjan2021/home>

⁸<https://www.bnl.gov/ompbrookathon2021/>

⁹<https://www.isc-hpc.com/schedule.html>

¹⁰<https://www.openmp.org/events/webinar-a-compilers-view-of-the-openmp-api/>

¹¹<https://www.openmp.org/events/webinar-the-leaders-of-openmp-discuss-the-future-of-the-openmp-api/>

8. “Introduction to OpenMP GPU Offloading¹²” by S. Pophale, R. Budiardja, and W. Elwasif September 22 - 23, 2021
9. ICPP’21 presentations:
 - “Loop Transformations Using Clang’s Abstract Syntax Tree”, LLVM in Parallel Processing Workshop, by M. Kruse
10. SC’21¹³
 - OpenMP Tutorials
 - “Advanced OpenMP: Host Performance and 5.1 Features” by C. Terboven, M. Klemm, R. van der Pas, B. R. de Supinski
 - OpenMP BoF
 - “OpenMP Offloading and the 5.2 API” session led by J. Doerfert and M. Klemm
 - OpenMP Booth Talks
 - “Behind the Pragmas” by J. Doerfert
 - “Low-overhead Loop Scheduling in OpenMP” by V. Kale
 - “Using OpenMP Loop Transformations with Clang” by M. Kruse
 - “SOLLVE Validation and Verification Suite” by S. Pophale
 - Workshops
 - “Loop Transformations Using Clang’s Abstract Syntax Tree”, LLVM-in-HPC Workshop, by M. Kruse

4.2.27 WBS 2.3.2.12 Flang

Overview The Flang project provides an open-source Fortran standard [148, 149, 150] compiler front-end for the LLVM Compiler Infrastructure (see <http://llvm.org>) [151]. Flang was formally accepted as an official component of LLVM in 2019 and merged portions of its initial code base into the main LLVM repository in April 2020. Work continues today with a growing set of contributors to the code base. Leveraging LLVM, Flang will provide a cross-platform Fortran solution available to ECP and the broader international LLVM community. The goals of the project include extending support to GPU accelerators and target Exascale systems, and supporting LLVM-based software and tools of interest to a large deployed base of Fortran applications.

LLVM’s growing popularity and wide adoption make it an integral part of the modern software ecosystem. Flang will provide a foundation for Fortran that will complement and interoperate with the Clang C/C++ compiler and other tools within the LLVM infrastructure. We aim to provide a modern, open-source Fortran implementation that is stable, has an active footprint within the LLVM community, and will meet the specific needs of ECP as well as the broader scientific computing community.

Key Challenges Today there are several commercially-supported Fortran compilers, typically available from one vendor and often for a limited set of platforms. None of these are open source. While the GNU gfortran compiler is open source and available on a wide variety of platforms, the source base is not modern LLVM-style C++ and the GPL license is not compatible with LLVM. This places limits on a wide breath of potential collaborations, and thus has an impact on broader community participation and adoption.

The primary challenge of this project is to create a source base with the maturity, features, and performance of proprietary solutions with the cross-platform capability of GNU compilers, and which is licensed and coded in a style that will be embraced by the LLVM community. Additional challenges come from supporting all Fortran language features, language extensions (e.g., OpenMP, OpenACC), and scalability required for effective use of exascale-class systems.

¹²<https://www.olcf.ornl.gov/calendar/introduction-to-openmp-gpu-offloading/>

¹³<https://sc21.supercomputing.org/program/>

Solution Strategy With the adoption of Flang into the LLVM community, our strategy focuses on establishing and growing a strong community around it and the development and delivery of a solid, alternative Fortran compiler for DOE’s platforms. It is critical that we be good shepherds within the LLVM community to successfully establish this community for Flang. This external engagement is in the best interest of ECP as well as the long-term success of Fortran, and enables leveraging the significant momentum and strengths of this widely used and accepted code base.

Our path to success will rely on significant testing across not only the various facilities but also across a very broad and diverse set of applications. Given the early development stage of Flang, this testing will be paramount in the delivery of a robust infrastructure to ECP and the broader community.

Recent Progress After several years of effort and support from NNSA, Flang was successfully adopted by the LLVM community and has transitioned from a stand-alone git repository to one hosted by the main LLVM project. This represents a significant result and the community reviewed and accepted code is available via GitHub: <https://github.com/llvm/llvm-project/>.

The current capabilities of Flang include the parsing and semantic analysis of the full Fortran 2018 standard and OpenMP 5.x. As part of the development of the parsing and semantic analysis portions of the front-end, over five million lines of Fortran code have been successfully processed. Beyond parsing and semantics, we have been focusing our efforts on creating a Fortran-centric intermediate representation (Fortran IR [FIR]) that leverages recent activities within Google on <https://www.blog.google/technology/ai/mlir-accelerating-ai-open-source-infrastructure/> (MLIR) for use with the implementation of FIR. With the development of FIR progressing, we have completed the first full (sequential) compiler with the full set of F77 capabilities, and will soon complete the full set of F95 capabilities. Implementations of the later standards will follow in chronological order.

Next Steps Our short-term priorities are focused on up-streaming of the sequential compiler with Fortran 95 support, completing implementation of the remaining Fortran standards, the creation of a significant testing infrastructure, and helping to play a key role in the interactions and overall discussions within the LLVM community. Longer term efforts will shift to support OpenMP 5.x features critical to ECP applications on the target Exascale platforms. We are actively exploring finding a common leverage point between Clang’s current OpenMP code base and Flang. This would enable the reuse of existing code versus writing everything from scratch in Flang. We see this as a critical path forward to enabling a timely release of a node-level parallelizing compiler for ECP. Additional work will focus on features that would benefit Fortran within the LLVM infrastructure as well as general and targeted optimization and analysis capabilities.

4.3 WBS 2.3.3 MATHEMATICAL LIBRARIES

End State: Mathematical libraries that (i) interoperate with the ECP software stack; (ii) are incorporated into the ECP applications; and (iii) provide scalable, resilient numerical algorithms that facilitate efficient simulations on Exascale computers.

4.3.1 *Scope and Requirements*

Software libraries are powerful means of sharing verified, optimized algorithms and their implementations. Applied research, development, and support are needed to extend existing DOE mathematical software libraries to make better use of Exascale architectural features. DOE-supported libraries encapsulate the latest results from mathematics and computer science R&D; many DOE mission-critical applications rely on these numerical libraries and frameworks to incorporate the most advanced technologies available.

The Mathematical Libraries effort will ensure the healthy functionality of the numerical software libraries on which the ECP applications will depend. The DOE mathematical software libraries used by computational science and engineering applications span the range from light-weight collections of subroutines with simple APIs to more end-to-end integrated environments and provide access to a wide range of algorithms for complex problems.

Advances in mathematical and scientific libraries will be necessary to enable computational science on Exascale systems. Exascale computing promises not only to provide more computational resources enabling

higher-fidelity simulations and more demanding studies but also to enable the community to pose new scientific questions. Exascale architectural characteristics introduce new features that algorithms and their implementations will need to address in order to be scalable, efficient, and robust. As a result, it will be necessary to conduct research and development to rethink, reformulate, and develop existing and new methods and deploy them in libraries that can be used by applications to deliver more complete and sophisticated models and provide enhanced predictive simulation and analysis capabilities.

The Mathematical Libraries effort must (1) collaborate closely with the AD effort (WBS 2.2) to be responsive to the needs of the applications and (2) collaborate with the other products within the ST effort (WBS 2.3) in order to incorporate new technologies and to provide requirements. All software developed within the Mathematical Libraries effort must conform to best practices in software engineering, which will be formulated early in the project in collaboration with the AD focus area. Software produced by this effort must provide scalable numerical algorithms that enable the application efforts to reach their performance goals, encapsulated in libraries whose data structures and routines can be used to build application software.

4.3.2 Assumptions and Feasibility

Years of DOE investment have led to a diverse and complementary collection of mathematical software, including AMReX, Chombo, Dakota, DTK, hypre, MAGMA, MFEM, PETSc/TAO, PLASMA, ScaLAPACK, SUNDIALS, SuperLU, and Trilinos. This effort is evolving a subset of existing libraries to be performant on Exascale architectures. In addition, research and development is needed into new algorithms whose benefits may be seen only at the extreme scale. Results of preliminary R&D projects indicate that this approach is feasible. Additionally, ECP will need to rely on a strong, diverse, and persistent base math research program, which is assumed to continue being supported by the DOE-SC ASCR Office.

4.3.3 Objectives

The high-level objective of the Mathematical Libraries effort is to provide scalable, resilient numerical algorithms that facilitate efficient application simulations on Exascale computers. To the greatest extent possible, this objective should be accomplished by preserving the existing capabilities in mathematical software while evolving the implementations to run effectively on the Exascale systems and adding new capabilities that may be needed by Exascale applications.

The key performance metrics for the software developed by this effort are scalability, efficiency, and resilience. As a result of the new capabilities in mathematics libraries developed under this effort, applications will tackle problems that were previously intractable and will model phenomena in physical regimes that were previously unreachable.

4.3.4 Plan

As detailed below, the Mathematical Libraries effort supports seven complementary L4 projects as needed to meet the needs of ECP applications. These efforts include strong collaborations among DOE labs, academia, industry, and other organizations, and leveraging existing libraries that are widely used by the DOE HPC community.

Initial efforts have focused on identifying core capabilities needed by selected ECP applications, establishing performance baselines of existing implementations on available Petascale and prototype systems, and beginning re-implementation of lower-level capabilities of the libraries and frameworks. Another key activity is collaborating across all projects in the Mathematical Libraries effort to define community policies in order to enable compatibility among complementary software and to provide a foundation for future work on deeper levels of interoperability. Refactoring of higher-level capabilities will be prioritized based on needs of the applications. In time, these efforts will provide demonstrations of parallel performance of algorithms from the mathematical software on pre-Exascale, leadership-class machines (at first on test problems, but eventually in actual applications). The initial efforts also are informing research into advanced exascale-specific numerical algorithms that will be implemented within the libraries and frameworks. In FY22–23, the focus will be on development and tuning for the specific architectures of the selected exascale platforms, in addition to tuning specific features that are critical to ECP applications. The projects will implement their software on the CORAL, NERSC and ACES systems, the pre-Exascale hardwares such as Spock and Yarrow, and

ultimately on initial Exascale systems, so that functionality, performance, and robustness can be evaluated by the applications teams and other elements of the software stack. Throughout the effort the applications teams and other elements of the software stack will evaluate and provide feedback on their functionality, performance, and robustness. These goals will be evaluated at least yearly based on milestones as well as joint milestone activities shared across the associated software stack activities by AD and HI project focus areas.

4.3.5 Risks and Mitigation Strategies

There are a number of foreseeable risks associated with the Mathematical Libraries effort.

Efficient implementation of new or refactored algorithms to meet Exascale computing requirements may introduce unanticipated requirements on programming environments. To mitigate this risk, effective communication is needed between projects in the Mathematical Libraries effort and projects tasked with developing the programming environments. From the application perspective, this is specifically tracked in a specific AD risk in the risk register. Additionally, the risks of an inadequate programming environment overall are tracked as a specific ST risk in the risk register.

A significant number of existing algorithms currently implemented in numerical libraries may scale poorly, thereby requiring significantly more effort than refactoring. The R&D planned for the first three years of the ECP is the first mitigation for this risk (as well as the codesign centers planned in AD). In addition, the ECP will be able to draw from a strong, diverse, well-run, persistent base math research program. From the application perspective, this is tracked via an AD risk in the risk register. Scaling issues for the software stack in general, including libraries, are monitored via an ST risk in the risk register.

Exascale architecture characteristics may force a much tighter coupling among the models, discretizations, and solvers employed, causing general-purpose solvers to be too inefficient. The mitigation strategy is to ensure close collaboration with the sub-elements of the AD focus area (WBS 2.2) to understand integration and coupling issues. Again, a strong, diverse, well-run, persistent base math research program may provide risk mitigation strategies.

4.3.6 Future Trends

Mathematical libraries have been one of the strongest success stories in the scientific software ecosystem. These libraries encode specialized algorithms on advanced computers that can be the difference between success or not. Algorithms such as multigrid, highly-tuned dense linear algebra and optimized FFTs, can improve performance by orders of magnitude and reduce the asymptotic algorithmic complexity for users. We foresee that math libraries will have an ever-growing role in the scientific software ecosystem, as architectures become more challenging for targeting optimization and algorithms require even more concurrency and latency hiding in order to realize performance on modern computer systems.

We anticipate that new algorithms based on multi-precision arithmetic will further enable performance improvements on compute devices that are optimized for machine learning workloads, where lower precision can be an order of magnitude faster than double precision. A recent paper [152] surveys the landscape of multi-precision numerical linear algebra algorithms. We are actively developing a number of algorithms outlined in that paper to take advantage of hardware speed with lower precision.

Over the course of the development of the xSDK libraries and interactions with the ECP applications teams, the need for batched sparse linear algebra (LA) functions has emerged in order to make more efficient use of the GPUs for many small and sparse linear algebra problems. The lack of the broad batched sparse LA operations will hinder several AD teams to reach their KPP goals. Therefore, in late FY21, we started a new activity on batched sparse solvers. We are collaborating with ECP industry partners to develop a set of new APIs to support required batched sparse LA operations for various GPUs. We will then develop the batched sparse linear solvers, both direct and iterative, and preconditioners on the GPUs. We will also develop new interoperability layers for integration across the applications, solvers, and lower level libraries. The planned work will involve the following math libraries: Ginkgo, hypre, Kokkos Kernels, MAGMA, PETSc, STRUMPACK, SUNDIALS, SuperLU, and Trilinos.

For a deeper discussion of the futures of ECP Math Libraries efforts, please consult the paper “Preparing Sparse Solvers for Exascale Computing” [153].

4.3.7 WBS 2.3.3.01 xSDK

Overview The xSDK project is creating a value-added aggregation of DOE math and scientific libraries through the xSDK [154], which increases the combined usability, standardization, and interoperability of these libraries as needed by ECP. The project focuses on community development and a commitment to combined success via quality improvement policies, better build infrastructure, and the ability to use diverse, independently developed xSDK libraries in combination to solve large-scale multiphysics and multiscale problems. We are extending xSDK package community policies and developing interoperability layers among numerical libraries in order to improve code quality, access, usability, interoperability, and sustainability. Focus areas are (1) coordinated use of on-node resources, (2) integrated execution (control inversion and adaptive execution strategies), and (3) coordinated and sustainable documentation, testing, packaging, and deployment.

xSDK is needed for ECP because it enables applications such as ExaAM and ExaWind to seamlessly leverage the entire scientific libraries ecosystem. For example, ExaWind has extremely challenging linear solver scaling problems. xSDK provides access to all scalable linear solvers with minimal changes. xSDK is also an essential element of the product release process for ECP ST. xSDK provides an aggregate build and install capability for all ECP math libraries that supports hierarchical, modular installation of ECP software. Finally, xSDK provides a forum for collaborative math library development, helping independent teams to accelerate adoption of best practices, enabling interoperability of independently developed libraries and improving developer productivity and sustainability of the ECP ST software products.

Key Challenges The complexity of application codes is steadily increasing due to more sophisticated scientific models. While some application areas will use Exascale platforms for higher fidelity, many are using the extra computing capability for increased coupling of scales and physics. Without coordination, this situation leads to difficulties when building application codes that use 8 or 10 different libraries, which in turn might require additional libraries or even different versions of the same libraries.

The xSDK represents a different approach to coordinating library development and deployment. Prior to the xSDK, scientific software packages were cohesive with a single team effort, but not across these efforts. The xSDK goes a step further by developing community policies followed by each independent library included in the xSDK. This policy-driven, coordinated approach enables independent development that still results in compatible and composable capabilities.

Solution Strategy The xSDK effort has two primary thrusts:

1. Increased interoperability: xSDK packages can be built with a single Spack package target. Furthermore, services from one package are accessible to another package.
2. Increased use of common best practices: The xSDK has a collection of community policies that set expectations for a package, from best design practices to common look-and-feel.

xSDK interoperability efforts began first with eliminating incompatibilities that prohibited correct compilation and integration of the independently developed libraries. These issues include being able to use a common version of a library by another library. The second, and ongoing phase is increased use of one package's capabilities from another. xSDK community package policies [3, 155] are a set of minimum requirements (including topics of configuring, installing, testing, MPI usage, portability, contact and version information, open source licensing, namespacing, and repository access) that a software package must satisfy in order to be considered xSDK compatible. The designation of xSDK compatibility informs potential users that a package can be easily used with others and makes configuration and installation of xSDK software and other HPC packages as efficient as possible on common platforms, including standard Linux distributions and Mac OS X, as well as on target machines currently available at DOE computing facilities (ALCF, NERSC, and OLCF) and eventually on new Exascale platforms. Community policies for the xSDK promote long-term sustainability and interoperability among packages, as a foundation for supporting complex multiphysics and multiscale ECP applications. In addition, because new xSDK packages will follow the same standard, installation software and package managers (for example, Spack [1]) can easily be extended to install many packages automatically.

For the adaptive execution effort, the team is working toward GPTune, a Gaussian process tuner, to help math library users find the optimal parameter settings for the libraries to achieve high performance for their applications. In addition, an interface will be created to also give access to alternate autotuners.

Another effort in the project is the development of a software quality toolkit that automates analyses and activities related to code testing, documentation and use. The goal is to facilitate the use of both general-purpose and custom code analysis tools to improve the quality of the xSDK libraries, and, more broadly, software within the high-performance applications community.

Recent Progress The xSDK team released xSDK 0.6.0 with two new xSDK libraries heFFTe and SLATE and is preparing xSDK 0.7.0 with planned addition of ArborX. Figure 55 illustrates the dependency graph of Spack-enabled interoperabilities of xSDK libraries in version 0.6.0 and the growth of the xSDK over the years. The team also looked for ways to increase interoperability within the xSDK. To achieve this goal, package

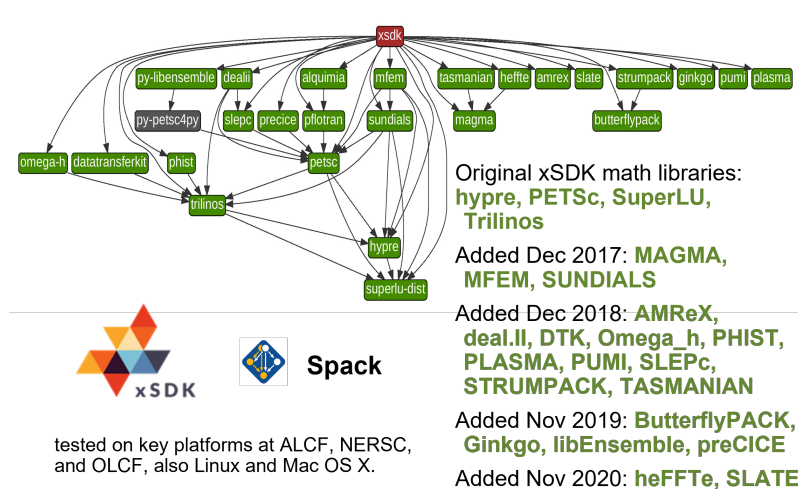


Figure 55: Dependency graph of xSDK packages with Spack-enabled interoperabilities represented in xSDK 0.6.0. A→B indicates A uses B.

developers were surveyed for opportunities to increase interoperabilities and a report was provided summarizing survey results and new and planned interoperability capabilities between xSDK members. Version 0.2.0 of the xsdk-examples code suite was released with updated and expanded examples to demonstrate xSDK library interoperabilities, including some CUDA-enabled examples.

Version 2.0 of the GPTune auto-tuning software for parameter optimization of HPC codes [156] was released. In this new release, the team finished the workflow design, with history database, shared repository and web interface for crowd-tuning, and developed the multi-fidelity GPTuneBand, combining LCM multitask learning and multi-armed bandit. The GPTuneBand tuning algorithm was applied to the electromagnetic equations solvers in the finite element xSDK library MFEM, and achieved almost 2x speedup compared to the default parameter settings.

The first version, v1.0.0, of a code quality toolkit, which contains open-source static and dynamic code analysis tools for C and C++, was released and applied to the xSDK libraries PETSc, SLEPc, SUPERLU and hypr. Example codes are provided in the xsdk-code-quality repository [157].

Early Access System Experiences The team has built subsets of the xSDK on Spock both with and without rocm, however the build requires special care described in the installation section of the xSDK website [154]. Efforts to extend the number of libraries and to create special instructions for Crusher are underway.

Next Steps Our next efforts include developing a plan to improve xSDK testing with an aim of sustainability; updating and enhancing the xsdk-examples test suite; implementing the xSDK testing plan; and releasing a

new xSDK with additional libraries and build options for GPUs, including software, documentation and user engagement materials.

4.3.8 WBS 2.3.3.01 xSDK Sub-project: *multiprecision*

Overview The multiprecision focus effort is a cross-laboratory effort to develop and deploy production-ready mixed precision algorithms for modern hardware architectures. The focus effort is ECP’s response to two trends we see in the evolution of HPC hardware: 1) An increasing number of processor designs feature low precision special function units to accommodate the demand of the Machine Learning community for high compute power in low precision formats; 2) The gap between compute power on the one hand and memory bandwidth on the other hand keeps increasing, making data access and communication prohibitively expensive compared to arithmetic operations. To address these trends, scientists from ECP partners teamed up to develop strategies and deploy production-ready software that reflects the architecture trends in the algorithm design. The multiprecision focus team has bi-weekly virtual meetings in which the progress on the different efforts is presented and discussed. As another integral part of the bi-weekly phone calls, we established a series of short talks where each meeting is commenced by an invited talk presenting an idea, success story, or progress update on mixed precision functionality to the audience. The short talks have quickly become a line-up of well-known researchers from ECP, the global academic HPC community, and industry.

Key Challenges Generally, there exists a strong relationship between the precision used in arithmetic operations and the accuracy of the computed result. Since scientific applications need to provide high-quality output, replacing high precision formats with low precision formats throughout a complete application code is generally not feasible. Instead, to utilize lower precision formats, the underlying numerical algorithms have to be redesigned to employ low precision formats for the most time-consuming parts while preserving high accuracy in the solution. In this context, arithmetic operations are only one aspect. As the execution time of many scientific applications is dominated by communication and memory access, the algorithm redesign also has to include strategies for compressing data to reduce the pressure on the memory bandwidth. This aspect becomes even more relevant as the arithmetic power continues to grow faster than the memory bandwidth, therewith widening the gap between arithmetic performance and memory performance, see Figure 56.

Solution Strategy In the multiprecision focus effort, we identified several action items. These include the development of

- a **memory accessor** that separates the precision format used in the arithmetic operations from the precision format used for memory operations and communication. This strategy can increase the accuracy of memory-bound low precision algorithms without impacting the performance, or increase the performance for memory-bound high precision algorithms that can tolerate or compensate for some information loss in the memory operations,
- **low precision kernels** using the reduced precision formats introduced for the machine learning community,
- **mixed-precision multigrid methods**,
- production-ready **mixed precision iterative refinement** variants for dense and sparse direct solvers,
- **mixed precision Krylov** solvers that accelerate the solution process by using a lower precision format for parts of the computations,
- **mixed precision preconditioners** that preserve the preconditioner quality but reduce the preconditioner application cost, and
- **mixed-precision FFT** algorithms.

Furthermore, to improve mixed precision interoperability, we investigate how to allow users to combine components from different xSDK libraries in different precision formats.

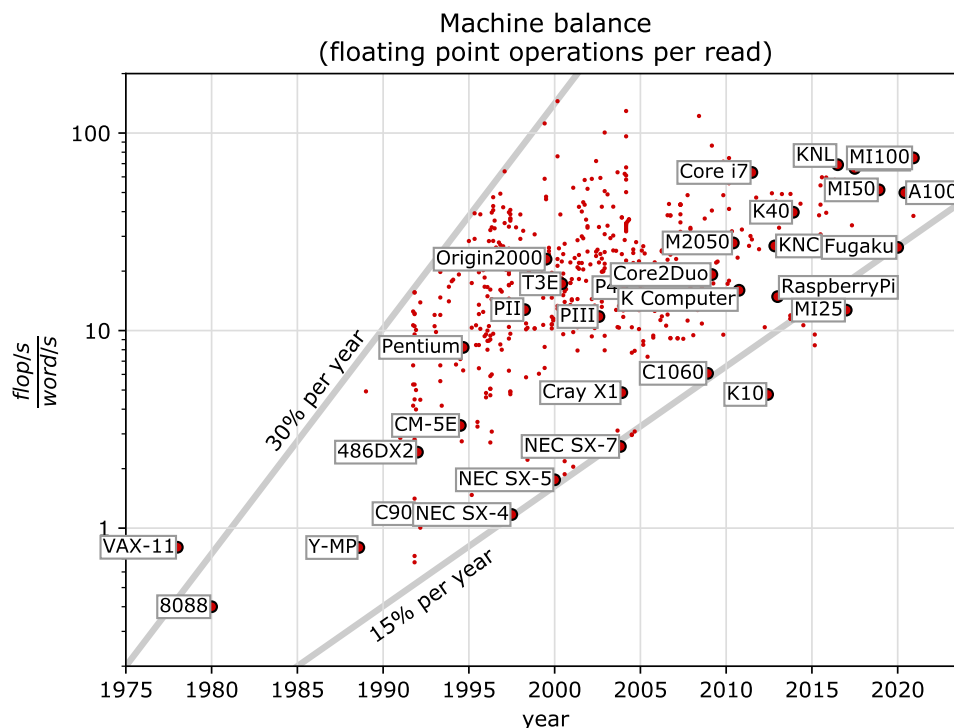


Figure 56: Evolution of the machine balance of processors over different hardware generations.

Recent Progress First, we deployed accessor-BLAS functionality for CPUs, AMD GPUs, and NVIDIA GPUs based on the memory accessor. This functionality is equivalent to memory-bound, low-precision BLAS, achieves the same performance, but provides higher numerical accuracy because all arithmetic operations are handled in high precision. The `dot`, `gemv`, and `trsv` kernels are now supported on NVIDIA GPUs.

Additionally, a Compressed Basis (CB-) GMRES solver was developed that can solve linear problems faster than the standard GMRES solver by storing the Krylov basis vectors in lower precision, thereby reducing the main memory access. This CB-GMRES solver is a plug-in replacement for standard GMRES because it achieves the same solution approximation accuracy, and all conversion and data compression are handled by the memory accessor and thus hidden from the user.

Furthermore, a production-ready, mixed-precision iterative refinement variant of the GMRES solver that can accelerate time-to-solution was deployed along with the prototype implementation of a distributed mixed-precision iterative refinement solver based on a sparse factorization. A production-ready, LU-based mixed-precision iterative refinement solver was deployed for AMD GPU architectures. Finally, FFT algorithms that use lower precision for internode communication were studied.

For a more comprehensive overview of the recent progress, the multiprecision focus team created an ECP report on “Advances in Mixed Precision Algorithms: 2021 Edition” that is available to the ECP community in Confluence. A significant portion of the material is currently under consideration for journal publication.

Next Steps Our next efforts include the publication of a compacted version of the ECP report on advances in mixed precision algorithms, the deployment of accessor-BLAS for AMD GPUs and multicore CPUs, and the advancement of multiprecision capabilities for solvers, preconditioners, and other ECP-relevant kernels in xSDK libraries.

4.3.9 WBS 2.3.3.01 xSDK Sub-project: batched sparse linear algebra

Overview Over the course of the development of the xSDK libraries and interactions with the ECP applications teams, the need for batched sparse linear algebra (LA) functions has emerged in order to make more efficient use of the GPUs for many small and sparse linear algebra problems. Currently there is almost no support from the vendors for batched sparse linear algebra routines, except for a few functions from NVIDIA (batched SpMV and batched sparse QR). The broad appeal of such a functionality that could be integrated across xSDK libraries represents a unique opportunity in terms of avoiding critical performance bottlenecks and integrating algorithmic advances across low-level software libraries used in ECP. We have identified six ECP applications that will benefit from new batched sparse solvers. These include chemical and nuclear kinetics, device modeling power grid planning and additive manufacturing applications. We will work with vendors to develop batched sparse linear algebra software, starting with the design of an interface, followed by the implementation of the software and their integration into application codes and libraries. We will evaluate the performance, pinpoint any issues and implement solutions to improve performance.

Key Challenges The applications we identified need to solve many small linear systems with varying sizes and sparsity patterns simultaneously. System sizes range between 30 and 200. The insight into the challenges of this effort then are the combination of dealing with sparsity, batched interface, iterative solvers, and matrix properties that depend on the elements' values. One of the main challenges facing this effort is how recent the discovery of the need for better handling of batched sparse LA is. This will require bridging the gap between what is possible on the upcoming Exascale hardware platform and what the eventual need of ECP applications will be. There are no such solvers available yet to draw the design potential and limitations from or try to conduct early feasibility and performance studies. This implies that a lot of the effort will face the lack of easy starting points and more careful analysis is needed on how to proceed. We envision challenges due to the structural sparsity that is already an issue on the accelerators for the existing software that deals with sparsity such as direct or iterative solvers and graph processing kernels. Additionally, potential solutions would be limited to what the application data allows. In particular, the matrix elements will determine numerical properties such as conditioning that could vary even if other parameters remain the same across the batch. All the variable factors mentioned above will require exploration and pose a challenge to limit the potential splintering of work with little eventual return. Minimizing this will require tight coordination between participants.

Solution Strategy We will try to draw on the existing libraries and their solvers and either extend their capability to our new context or limit their features that are an unlikely fit for the specialized function of batched sparse LA for small batch sizes, such as multi-GPU support. With the abundance of iterative solvers, we will start exploring their functionality and potential extensions for our effort. We are collaborating with ECP industry partners, including NVIDIA, AMD, and Intel, to develop a set of new APIs to support required batched sparse linear algebra operations for various GPUs. This is essential to receive an early and frequent feedback on the direction and the potential benefit of the solvers and data formats we propose and evaluate. We will then develop batched sparse linear solvers, both direct and iterative, and preconditioners for the GPUs. We will also develop new interoperability layers for integration across the applications, solvers, and lower-level libraries. The planned work will involve the following math libraries: Ginkgo, hypre, Kokkos Kernels, MAGMA, PETSc, PLASMA, STRUMPACK, SUNDIALS, SuperLU-DIST, and Trilinos. We envision that the functionality should be for on-device compute with careful use of shared/local/private memory (e.g., only 64KB on NVIDIA GPU). The interface will support multiple sparse matrix formats. For language interoperability, C++ and C will be the core languages, and SWIG will be used to build the Fortran wrappers.

Recent Progress In initial meetings with team members and vendors, we started reviewing existing batched interface options in vendor and xSDK libraries. We extracted application data in the form of small sparse matrices that occur in batches in the Chemical kinetics ECP application Pele. They are now available to the project members in an easily readable form for analysis and testing with the existing and future batched kernels. We started reviewing the options available for host-side and device-side interface design.

Next Steps Our next efforts include the finalization of vendor batched interface review, the organization of batched sparse matrix examples, the design of batched sparse linear algebra interface for the relevant application use cases, the initial implementation of batched sparse linear algebra software, and the integration of batched sparse LA software into applications and libraries and performance assessment of effort.

4.3.10 WBS 2.3.3.06 PETSc-TAO

Overview Algebraic solvers (generally nonlinear solvers that use sparse linear solvers) and integrators form the core computation of many numerical simulations. No scalable black box sparse solvers or integrators work for all applications, nor are there single implementations that work well for all problem sizes. Hence, algebraic solver and integrator packages provide a wide variety of algorithms and implementations that can be customized for the application and range of problem sizes. PETSc/TAO [158, 159] is a widely used numerical library for the scalable solution of linear, nonlinear, and variational systems, for integration of ODE/DAE systems and computation of their adjoints, and for numerical optimization. This project focuses on three topics: (1) partially matrix-free scalable solvers to efficiently use many-core and GPU-based systems; (2) reduced synchronization algorithms that can scale to larger concurrency than solvers with synchronization points; and (3) performance and data structure optimizations for all the core data structures to better utilize many-core and GPU-based systems as well as provide scalability to the exascale systems.

The availability of systems with over 100 times the processing power of today’s machines compels the utilization of these systems not just for a single forward solve (as discussed above), but rather within a tight loop of optimization, sensitivity analysis (SA), and uncertain quantification (UQ). This requires the implementation of a new scalable library for managing a dynamic hierarchical collection of running scalable simulations, where the simulations directly feed results into the optimization, SA, and UQ solvers. This library, which we call libEnsemble, directs the multiple concurrent function evaluations through the tight coupling and feedback. This work consist of two parts: (1) the development of libEnsemble; and (2) the development of application-relevant algorithms to utilize libEnsemble.

Key Challenges A key challenge for scaling the PETSc/TAO numerical libraries to Exascale systems is that traditional sparse-matrix-based techniques for linear, nonlinear, and ODE solvers, as well as optimization algorithms, are memory-bandwidth limited. Another difficulty is that any synchronizations required across all compute units—for example, an inner product or a norm—can dramatically affect the scaling of the solvers. Another challenge is the need to support the variety of accelerators that will be available on the exascale systems and the programming models that application teams use for performance portability.

Running an ensemble of simulations requires a coordination layer that handles load balancing and allows the collection of running simulations to grow and shrink based on feedback. Thus, our libEnsemble library must be able to dynamically start simulations with different parameters, resume simulations to obtain more accurate results, prune running simulations that the solvers determine can no longer provide useful information, monitor the progress of the simulations, and stop failed or hung simulations, and collect data from the individual simulations both while they are running and at the end.

Solution Strategy To address the scalability of the numerical libraries, we implemented new solvers and data structures including: pipeline Krylov methods that delay the use of the results of inner products and norms, allowing overlapping of the reductions and other computation; partially matrix-free solvers using high-order methods that have high floating-point-to-memory-access ratios and good potential to use many-core and GPU-based systems; and in-node optimizations of sparse matrix-matrix products needed by algebraic multigrid to better utilize many-core systems.

Our strategy for coordinating ensemble computations has been to develop libEnsemble to satisfy our needs. This library should not be confused with workflow-based scripting systems; rather it is a library that, through the tight coupling and feedback, directs the multiple concurrent function evaluations needed by optimization, SA, and UQ solvers.

Accelerator Strategy Our overall strategy for accelerator support in PETSc/TAO is based on flexibility and a separation of concerns by wrapping the data pointers from the user programming language and programming model inside of PETSc Vector and Matrix vector objects. This approach allows us to focus our

Performance portability in PETSc

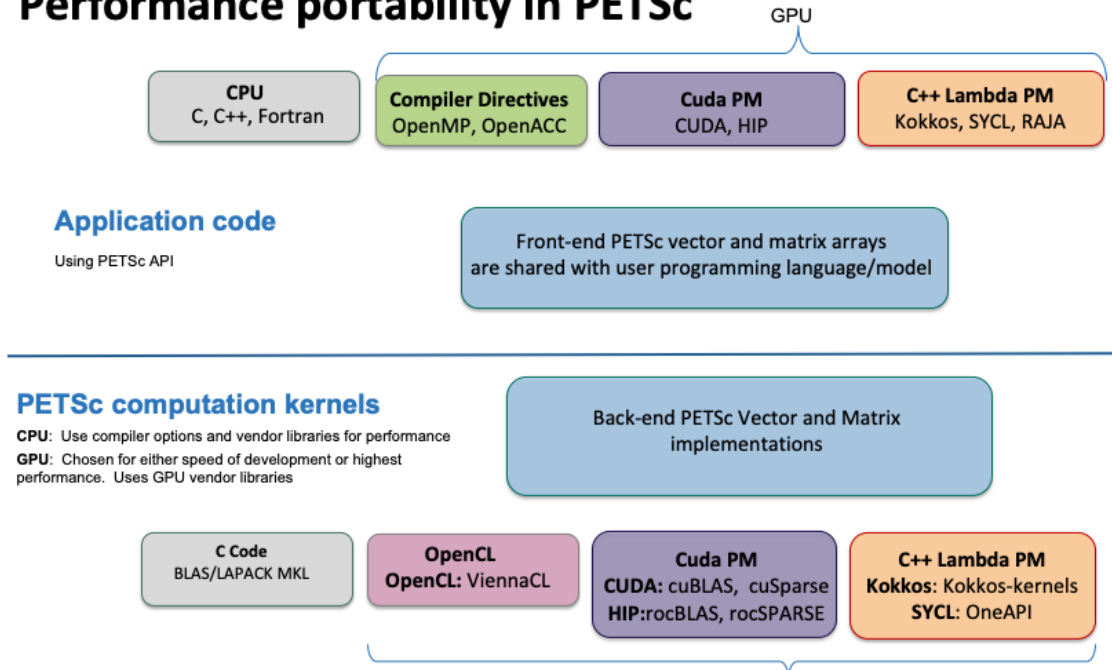


Figure 57: The PETSc/TAO architecture enables users to utilize a variety of programming models for GPUs independently of PETSc’s internal programming model.

effort on the kernels, such as vector, matrix-vector, matrix-matrix, and other fused computational kernels, while the developer can focus on their application. We provide multiple backends and support AMD, Intel, and NVIDIA accelerators. Thus, we support the tools that the application developers are using, while obtaining performance in the numerical libraries and kernels. The architecture can be found in Figure 57.

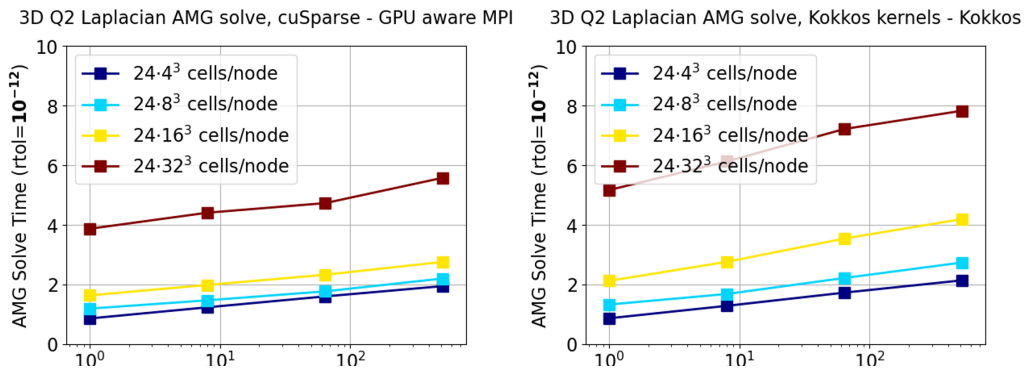


Figure 58: Solve time for a 3D Laplacian with second-order elements. Larger grids are generated by uniform refinement. Runs are configured with six resource sets on each Summit node, each with one GPU and 4 MPI processes.

Our primary performance portability layer is based on Kokkos and KokkosKernels, which supports vector, matrix, and matrix-matrix operations. We also have full support for a native CUDA backend and partial support for a native HIP and OpenCL backends. Figure 58 demonstrates performance portability via a scaling study with PETSc/TAO’s built-in algebraic multigrid (AMG) solver, PCGAMG, using cuSPARSE and Kokkos (with KokkosKernels) back-ends on our most mature device, CUDA, where we obtain competitive performance. The slower performance of KokkosKernels is due to computing a transpose for matrix transpose

multiply, which is not yet natively supported in KokkosKernels.

The PETSc/TAO library has been compiled for Spock and Crusher and preliminary results have been obtained using the accelerators and performance optimizations are being made.

The libEnsemble tools are based on python and makes calls to user-defined functions. The user-defined functions can use the accelerators on the compute nodes during their evaluation.

Recent Progress In the past year, we have released PETSc/TAO 3.16 (available at <http://www.mcs.anl.gov/petsc>), which features enhanced GPU support. We have full support for a Kokkos plus KokkosKernels backend for performance portability that includes vector, matrix, and matrix-matrix operations and full support for a native CUDA backend. We have partial support for a native HIP and OpenCL backends. We have been updating and profiling the GAMG solver, the native algebraic multigrid solver in PETSc, to use the enhanced GPU support. Numerical results on this work is available in [160]. We have also been updating our scalable communication layers, PetscSF, which allows us to use GPU-aware MPI, thus allowing direct communication of data between Summit GPUs, bypassing the previously needed step of first copying the data to the CPU memory. Numerical results on this work is available in [161].

We have also released libEnsemble 0.7.2 (available at <https://github.com/Libensemble/libensemble>). This release includes new generator functions and examples, improved testing across available platforms, and a new tutorial. A paper documenting libEnsemble and providing use cases is available in [162].

Next Steps The following efforts have been identified for the next phase of the project.

Application readiness We will complete a status review of our applications for the early access hardware. We will complete software quality initiatives related to the build system, architecture specific Spack recipes, and Spack smoke tests for build and accelerator usage validation.

PETSc/TAO + Kokkos + KokkosKernels release We will release a version of PETSc/TAO with full Kokkos and KokkosKernels integration. We will provide a tutorial on these features, characterize the performance, and suggest optimizations and best practices. We will also make improvements to the PETSc/TAO GAMG solver and provide updated performance results.

libEnsemble + Balsam2 release We will release a version of libEnsemble with full Balsam2 support and updated API. We will follow continuous integration best practices and continue testing on pre-exascale DOE systems.

PETSc/TAO + Optimized communication layer release We will release a version of PETSc/TAO with an optimized communication layer for the early access systems. This version will include support for AMGx on NVIDIA GPUs and a batch LU factorization and solver. We will optimize some numerical methods (such as methods in DMNetwork and numerical optimization methods) to use the communication layer and better utilize the accelerators. We will provide initial benchmark results for GAMG and AMGx.

4.3.11 WBS 2.3.3.07 STRUMPACK-SuperLU

Overview This project will deliver factorization-based sparse solvers encompassing the two widely used algorithm variants: supernodal (SuperLU: <https://portal.nersc.gov/project/sparse/superlu>) and multifrontal (STRUMPACK: <http://portal.nersc.gov/project/sparse/strumpack>). STRUMPACK is further enhanced with scalable preconditioning using hierarchical and data-sparse matrix algebra. Both libraries are purely algebraic, applicable to many application domains. We will address several Exascale challenges, with the following focus areas: (1) Develop novel approximation algorithms that have lower arithmetic and communication complexity with respect to the size of the input matrix; (2) Develop new parallelization strategies that reduce inter-process communication and expose task parallelism and vectorization for irregular computations involving sparse data structures to better use on-node resources; (3) Integrate our software into higher-level algebraic solvers such as hypre, PETSc, Trinos, and collaborate with ECP teams for application-specific and hardware-specific tuning of the parameters space to achieve optimal efficiency.

Our solver technology is essential for ECP, because many codes expected to run on Exascale machines need solutions of sparse algebraic systems, and many high-fidelity simulations involve large-scale multiphysics

and multiscale modeling problems that generate highly ill-conditioned and indefinite algebraic equations, for which pure iterative methods cannot converge to the solution. The factorization-based algorithms being developed herein represent an important class of methods that are indispensable building blocks for solving those numerically challenging problems. Our software can often be used as a reliable standalone solver, or as a preconditioner for Krylov methods, or as a coarse grid solver in multigrid methods.

Key Challenges At Exascale we need to address several major challenges: decreasing amount of memory per core, increasing impact of communication cost and load imbalance, and increasing architectural heterogeneity. Our new design of algorithms and codes must focus on reducing communication and synchronization and task scheduling instead of floating point operation throughput. In sparse factorization methods, we expect new bottlenecks in parts of the code that previously received little attention. For example, the preprocessing step involves numerical pivoting for selecting stable pivots and symbolic factorization, which do not yet parallelize well on manycore architectures with fine-grained parallelism. At Exascale, direct solvers are more likely to be used in a preconditioning strategy, for example, in block Jacobi preconditioning, in domain decomposition methods or as coarse-grid solvers in algebraic multigrid, which requires repeated triangular solves. The challenge here is to mitigate the low arithmetic intensity and high degree of data dependency.

Compared to iterative methods, the primary bottleneck of direct solvers is the asymptotically higher growth in memory need and floating point operations, especially for problems from three-dimensional geometry. It is imperative to develop new factorization methods that require much less memory and data movement.

Solution Strategy We will address these challenges in several thrust areas. The new techniques will be implemented in the two software packages SuperLU and STRUMPACK. The former is a widely used sparse direct solver based on supernodal factorization and the latter is a newer direct solver/preconditioner package based on multifrontal factorization and hierarchical low-rank matrix structures.

The improvements for SuperLU will be mainly in two areas: (1) develop the communication-avoiding 3D factorization and triangular solve algorithms and codes that have provably lower communication complexity; (2) develop a synchronization-avoiding triangular solve code to enable more overlap of communications of different processes at different substitution steps; (3) develop new multiGPU codes for both symbolic preprocessing step and numerical factorization and solve steps.

In addition to exploiting structural sparsity as SuperLU does, STRUMPACK also exploits data sparseness in the dense blocks of sparse factors using low-rank representations, which leads to linear scaling $O(n)$ or $O(n \log n)$ memory and arithmetic complexity for PDEs with smooth kernels. The developments for STRUMPACK will focus on several areas: (1) development of an advanced preconditioner based on approximate multifrontal LU factorization, combining small dense fronts, medium sized Block Low-Rank (BLR) fronts (reducing the complexity pre-factor) and large fronts compressed using the Hierarchically Off-Diagonal Butterfly (HODBF) representation (leading to nearly linear scaling for large high-frequency problems); (2) port the BLR algorithms – including several variants like left and right-looking, low-rank update and accumulate – to GPUs (targeting CUDA, HIP/ROCm and SYCL/DPC++), and include in the preconditioner; and (3) develop sparse direct solvers (preconditioners) that are fully resident on the GPU, including reordering and symbolic analysis.

Recent Progress We mainly focus on the multiGPU developments. For SuperLU, the new developments are on triangular solve with one-sided communication using Nvidia’s NVSHMEM and AMD’s ROC SHMEM (in design stage). For STRUMPACK we further improved performance of the sparse direct solver on GPUs, by reducing memory allocations. We reduced the peak memory usage of the BLR enabled preconditioner, allowing to solve much larger problems. We also worked with the ECP application ExaSGD team, applying our sparse solvers to the linear systems coming from AC optimal power flow problems. The linear systems arising from the Interior Point optimization loops are highly ill-conditioned, and zero-pivots are encountered during numerical factorization. We improved both STRUMPACK and SuperLU to deal with the situation and allow the factorization to succeed and recover the solution accuracy by iterative refinement.

The other algorithmic changes and the results are detailed below.

STRUMPACK We improved the block low-rank preconditioner to drastically reduce the peak memory usage. This is achieved by building the BLR representation one single column at a time. This has allowed us to solve

numerically challenging 3D problems up to 350^3 , using double precision complex arithmetic, on 64 nodes of Cori. We implemented different BLR variants, including left-looking (reducing communication), and low-rank update and accumulate operations (reducing algorithm complexity). In addition to NVIDIA and AMD GPU support [163], STRUMPACK now has experimental support for Intel GPUs through SYCL/DPC++ and the oneAPI standard’s variable sized batched BLAS and LAPACK routines.

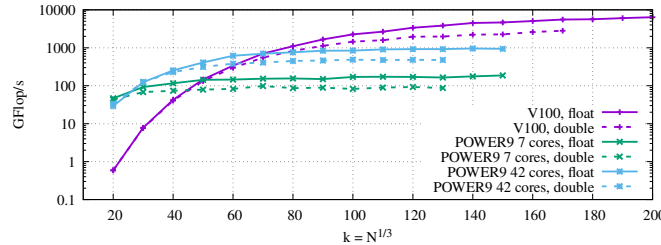


Figure 59: STRUMPACK numerical factorization performance on Summit.

SuperLU For the multiGPU sparse triangular solve (SpTRSV), we leverage the advantage of GPU-initiated data transfers of NVSHMEM. The new multiGPU SpTRSV implementation using two CUDA streams achieves a $3.7\times$ speedup when using twelve GPUs (two nodes of the Summit supercomputer at ORNL) relative to our implementation on a single GPU, and up to $6.1\times$ compared to NVIDIA’s cuSOLVER (cuSPARSE csrsv2) over the range of one to eighteen GPUs [164]. In the new v7.0.0 release of SuperLU_DIST, we released the 3D factorization algorithm, where MPI ranks are arranged as 3D process grid: $P_x \times P_y \times P_z$. P_z direction is used internally for storing sub-parts of Schur-complement updates. The input $\{A, B\}$ matrices and output X matrix need to reside on $P_x \times P_y$. We implemented a new interface that enables redistribution of the users matrices between all MPI ranks and the layer $P_x \times P_y$. The interface is flexible, allows uneven block partition of the user input, and allows separate calls to factorization and triangular solves.

Preliminary Experiences on Early Access Systems In the past year, we have ported both STRUMPACK and SuperLU to Spock using HIP/ROCm backend. For STRUMPACK, we observed that Spock single MI100 (ROCm 4.2) GPU is usually slower than Summit single V100 GPU; it can be $1.9\times$ slower. For SuperLU, the single GPU performance is comparable on Spock MI100 and Summit V100. Using 16 Spock GPUs, we observed $2\times$ speedup over CPU-only SuperLU code.

More recently, we started working on Crusher. The following are preliminary results. For both STRUMPACK and SuperLU, the Crusher single MI250X is consistently faster than the Spock MI100, up to 85%. We also ran SuperLU on multiple Crusher nodes, up to 8 nodes with 64 GPUs, and found that the parallel scaling is close to that on Spock.

Next Steps Our future efforts will focus on the following areas: For STRUMPACK, we will work to improve the performance of the block low-rank preconditioner on multiGPU systems. For SuperLU, we will develop a new GPU-friendly meta-data structure for the sparse U matrix, replacing the skyline data structure. That trades increased memory usage for parallelism-friendly memory access, and is will particularly helpful to improve U-solve performance on GPU. Furthermore, we will apply the GPTune autotuner developed from xSDK4ECP project to conduct comprehensive tuning in the parameter space for both STRUMPACK and SuperLU, for the ECP applications and on the pre-exascale machines.

4.3.12 WBS 2.3.3.12 Sub-project: SUNDIALS

Overview This project is enhancing the SUNDIALS library of numerical software packages for evolving differential systems in time using state-of-the-art time integration technologies for use on exascale systems.

The SUNDIALS suite of packages [165] provides efficient and adaptive time integrators and nonlinear solvers. The packages are written using encapsulation of both data structures and solvers, thus allowing easy incorporation into existing codes and flexibility to take advantage of new solver technologies and packages. SUNDIALS provides both multistep and multistage methods designed to evolve stiff or nonstiff ordinary

(ODE) and differential algebraic (DAE) systems with efficient accuracy-driven time step selection. SUNDIALS also provides both Newton and fixed point (with optional acceleration) nonlinear solvers and scaled Krylov methods with hooks for user-supplied preconditioners. SUNDIALS is released with data structures and interfaces to solvers supporting several programming environments, and users can employ these supplied structures and interfaces or provide their own data structures or solvers under the integrators.

Through software infrastructure developments, this project is enabling the efficient and robust SUNDIALS time integrator packages to easily interoperate with applications evolving time dependent systems as well as with external linear and nonlinear solver packages developed for exascale computing. In addition, this project is providing support for integrating several independent ordinary differential equation systems simultaneously on GPUs as part of multiphysics applications. Lastly, this project is supporting the deployment and use of SUNDIALS packages within ECP applications, both through incorporation into the discretization-based Co-Design Centers, AMReX and CEED, and directly into applications.

Key Challenges Current implementations of efficient time integrators face challenges on many fronts. First, applications need both efficient integrators and ones that can interface easily with efficient linear algebra packages to solve subservient linear systems. In addition, integrators and their interfaces to both solver libraries and applications must be frequently updated to keep up with rapid advances in system architectures. Some ECP applications require the solution of many small systems of ODEs in parallel on GPUs giving rise to the specific need for a GPU-enabled ODE time integration capability that can be used in parallel for many systems at once and be able to run on multiple GPU-based architectures with differing programming models. Lastly, ECP applications require assistance incorporating new linear solvers underneath the integrators and in updating their interfaces to optimally use integrators on new platforms.

Solution Strategy This year, the SUNDIALS team will be performing regular performance assessments of the SUNDIALS integrators within key ECP applications that use SUNDIALS, including PeleC, PeleLM, and Nyx. These assessments will involve running the application codes on available early access hardware and evaluating SUNDIALS performance, both due to systems aspects as well as algorithmic choices.

In addition, the SUNDIALS team will be working with the Pele team to assess a new interface to the Ginkgo library of sparse linear solvers that will allow the PeleLM and PelePhysics codes access to highly efficient solvers for batched systems that work well on AMD GPUs. This interface will provide additional options to those available in the MAGMA solver library. In addition, the SUNDIALS team will evaluate the application of scaling factors for Jacobian matrices supplied to direct linear solvers like MAGMA and Ginkgo. These factors have long been applied in the SUNDIALS use of iterative linear solvers, and they drastically improve the numerical conditioning of the linear systems.

Also this year, the SUNDIALS team will evaluate the inclusion of mixed precision linear solvers underneath its integrators. While higher precision has generally been necessary in time integrator algorithms due to the need to meet accuracy tolerances, the linear solvers are used solely to update the solutions to the nonlinear systems. Hence these linear solvers can be fairly imprecise. The team will explore using the newest mixed precision solvers as a way to help speed solutions on GPU-based systems.

Lastly, the SUNDIALS team is providing general support to ECP applications in interfacing SUNDIALS packages into their software and in the optimal use of advanced time integration algorithms. This support will include working with the application teams to help them install SUNDIALS, adjust their build systems to appropriately link with the SUNDIALS library, choose the best algorithms for their applications, and efficiently use (through parameter adjustments) the time integrator methods in SUNDIALS.

Recent Progress SUNDIALS had four releases this past year, including new features in direct support of ECP application needs. In particular, releases in late 2020 delivered support for vectors, matrices, and solvers that make use of the HIP programming model for use on AMD GPUs. In addition, releases in spring and summer of 2021 delivered similar support for the SYCL programming model for use on Intel GPUs. These new features are being used by the Pele and Nyx code teams, and performance assessments are ongoing.

To support better performance evaluation within ECP applications, in FY21, the SUNDIALS team implemented a new high performance test suite and performance assessment layer. The test suite consists of several short problems that can scale to high numbers of unknowns and make use of a variety of programming

models and hybrid distributed parallel/GPU systems. The test suite allows for easier evaluation of performance on new platforms, including expected exascale systems. Moreover, the SUNDIALS team added a performance assessment layer and enabled use of the ECP ST Caliper package for instrumentation underneath that layer. This year, the SUNDIALS team will conduct regular testing of this new suite on EAS systems as a way of regularly updating build and install options and improving library performance.

In fall of 2020, the SUNDIALS team completed initial assessments of performance on AMD GPU systems [166] showing $5\times$ speedups using MPI+HIP over MPI+serial on advection-reaction problems using some of the SUNDIALS advanced integrators. In spring of 2021, the SUNDIALS team completed a comprehensive study of the relative performance of several time integrators within the PeleC application. This study gave rise to new recommendations for parameter settings that led to a 44% reduction in runtime.

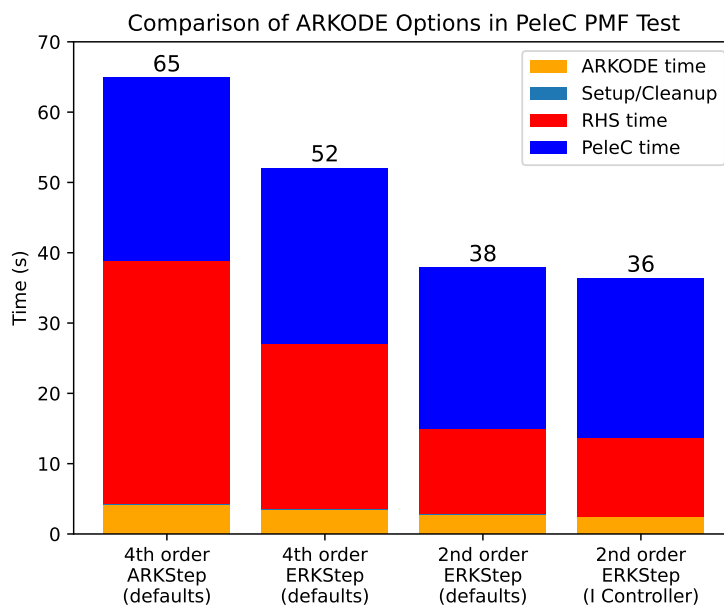


Figure 60: Runtimes from a pre-mixed flame test case in PeleC using various time integrators from SUNDIALS for chemistry evolution. Demonstrated 44% reduction in runtime through algorithm choice using 2 nodes and 12 GPUs on Summit. Note that more efficient algorithms required fewer calls to the expensive right-hand-side function.

Early Access System Experiences The SUNDIALS team has been testing performance on both the Spock Early Access System and on the new Crusher system. Figure 61 shows execution times for a 2D diffusion benchmark problem using the SUNDIALS CVODE integrator with a preconditioned conjugate gradient linear solver and run on one node of Spock using the new SUNDIALS profiling capabilities. The maximum speedups (cpu time / gpu time) achieved are $3.8\times$, $3.9\times$, $2.1\times$, and $7.8\times$ for the overall, linear solve, dot product, and right-hand-side times, respectively. Furthermore, application partners with the PeleC, PeleLMeX, PelePhysics, Nyx, and MEUMAPPS-SS codes have all been able to run SUNDIALS within their codes on Spock.

In addition, the SUNDIALS team recently ported to Crusher and executed the same 2D diffusion benchmark problem there using both the CVODE and ARKODE integrator packages. These results are in Figure 62. Running with problem sizes of 10^6 and 10^7 , speedups of $\approx 1.53\times$ for ARKODE and of $\approx 1.45\times$ for CVODE were observed with Crusher over Spock ($T_{Spock}/T_{Crusher}$). In addition, a speedup of $\approx 1.55\times$ was observed for just the dot product operation. These experiments used 1 MPI task with 1 MI100 GPU on Spock and 1 GCD of a MI250X GPU on Crusher. Lastly, we note that the MEUMAPPS-SS (ExaAM project), PeleLMeX, and Nyx codes have successfully run using SUNDIALS on Crusher.

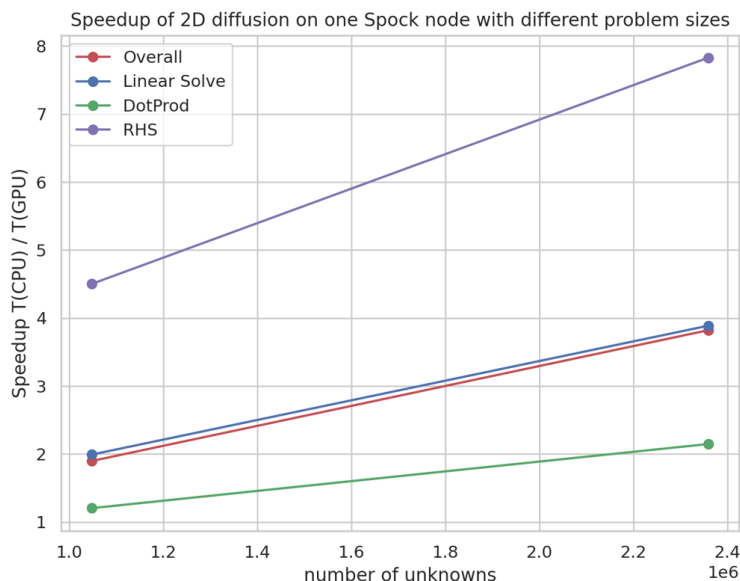


Figure 61: Speedup of 2D diffusion on one Spock node with differing problem sizes. CPU problems used 60 CPU cores and GPU problems used all 4 MI100 GPUs on the node.

Next Steps During the remainder of FY22, this project team will perform regular assessments of SUNDIALS performance with the new high-performance test suite and within key applications on Early Access Systems, implement new features to support more efficient linear solves, expand SUNDIALS support for Intel GPUs, and continue to support ECP applications in their use of SUNDIALS.

4.3.13 WBS 2.3.3.12 Sub-project: hypre

Overview The hypre software library [167, 168] provides high performance preconditioners and solvers for the solution of large sparse linear systems on massively parallel computers, with particular focus on algebraic multigrid solvers. One of hypre’s unique features is the provision of a (semi)-structured interface, in addition to a traditional linear-algebra based interface. The semi-structured interface is appropriate for applications whose grids are mostly structured, but with some unstructured features. Examples include block-structured grids, composite grids in structured adaptive mesh refinement (AMR) applications, and overset grids. These interfaces give application users a more natural means for describing their linear systems, and provide access to methods such as structured multigrid solvers, which can take advantage of the additional information beyond just the matrix. Since current architecture trends are favoring regular compute patterns to achieve high performance, the ability to express structure has become much more important. The hypre library provides both unstructured and structured multigrid solvers, which have shown excellent scalability on a variety of high performance computers, e.g Blue Gene systems (unstructured solver BoomerAMG has scaled up to 1.25 million MPI cores with a total of 4.5 million hardware threads). It is used by many ECP application teams, including ExaAM, Subsurface, ExaWind, CEED, and more. It requires a C compiler and an MPI implementation, but it also runs in an OpenMP environment. It also has GPU capabilities.

Key Challenges While hypre’s solvers contain much parallelism, their main focus is the solution of sparse linear systems, leading to very large demands on memory bandwidth. In addition, the use of multiple levels, while greatly aiding convergence of the solvers, leads to decreasing systems sizes, number of operations and parallel efficiencies on coarser levels. Particularly the unstructured algebraic multigrid solver BoomerAMG[169], which is hypre’s most often used preconditioner, suffers from increasing communication complexities on coarser levels. Coarse grid operators are generated by multiplying three matrices leading to increasing numbers of nonzeros per row in the resulting matrices and with it increasing numbers of neighbor

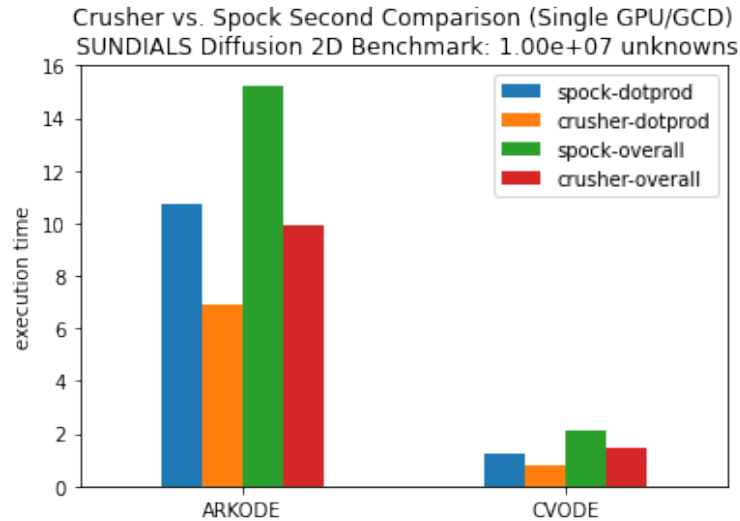


Figure 62: Speedup of 2D diffusion with 10^7 unknowns on 1 MPI task with 1 MI100 GPU on Spock and 1 GCD of a MI250X GPU on Crusher.

processes. While BoomerAMG’s solve phase mainly consists of matrix vector products and smoothing operations, which are fairly straight forward to parallelize, even on a GPU, its setup phase is highly complex, including many branches, a lot of integer operations as well as some sequential passages. Previous interpolation strategies that lead to best convergence and performance on distributed memory machines were not suitable for implementation on GPUs or similar architectures requiring extreme parallelism and required new algorithmic approaches. Since hypre is a mature product with many solvers and interdependent features, any significant changes that affect the whole library, are tedious and require much testing to ensure that the library stays backward compatible and no features are broken.

Solution Strategy Since computer architectures continue to change rapidly, it was important to come up with strategies that will facilitate future porting of the software. Therefore we developed and implemented a new memory model that addresses the use of different memory locations. Since the upcoming computer architectures are heterogeneous with accelerators, we focus on enabling hypre for GPUs. We have looked into various options, such as the use of CUDA, OpenMP 4.5, as well as RAJA and Kokkos. We limited the latter three options to the structured interface and solvers which are more natural candidates for such an approach due to their use of macros, called BoxLoops, for loops. We adopted a modular approach for the unstructured interface, which relies on the restructuring the solver components to use smaller kernels that are and/or will be implemented in CUDA for Nvidia GPUs. Since hip is similar to CUDA, porting to AMD GPUs has been fairly straight forward. We are now investigating the use of a vendor conversion tool from CUDA to SYCL to port structured and unstructured solvers to upcoming exascale computers with Intel GPUs.

Recent Progress Previously we had enabled the structured interface and solvers, SMG and PFMG[170], to completely run on GPUs, using CUDA, OpenMP4.5, RAJA and Kokkos, and have now also added hip to enable use on AMD GPUs. For our unstructured AMG solver BoomerAMG, we had implemented suitable CUDA kernels for setup and solve phase, designed a new class of interpolation operators based on sparse matrix operations[171], implemented it on GPUs and ported aggressive coarsening to the GPU. Recently, we added Umpire support for memory pooling on GPUs, which can significantly improve performance. We have enabled hypre’s specialized solvers, including linear and eigensolvers for Maxwell problems and solvers for H-div problems, to run on Nvidia GPUs. Figure 63 shows some results for a Maxwell problem using finite elements of increasing order on 1 node of Lassen. We have ported many of the CUDA kernels and routines in the unstructured interface to hip to enable their use on AMD GPUs. We have also developed a new version of multipass interpolation based on matrix-matrix multiplications and implemented it in CUDA

and hip. Multipass interpolation is a popular low-memory prolongation used with aggressive coarsening. Figure 64 shows a comparison of CPU and GPU runtimes on 2 nodes of Spock for a 3D diffusion problem with a 27-point stencil on a $n \times n \times n$ grid using AMG-PCG with aggressive coarsening on the first level for increasing n .

FEM order	System size	CPU time	GPU time	Speedup
1	92,256	0.22	0.87	0.25
2	712,896	3.68	2.03	1.81
3	2,378,016	31.15	7.03	4.43

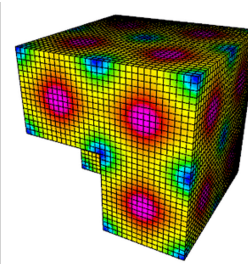


Figure 63: Solution of a simple 3D electromagnetic diffusion problem corresponding to the 2nd order definite Maxwell equation $\nabla \times \nabla \times E + E = f$ using AMS-PCG on 1 node of Lassen (4 GPUs vs. 40 CPU cores) using finite elements of increasing order on a Fichera mesh.

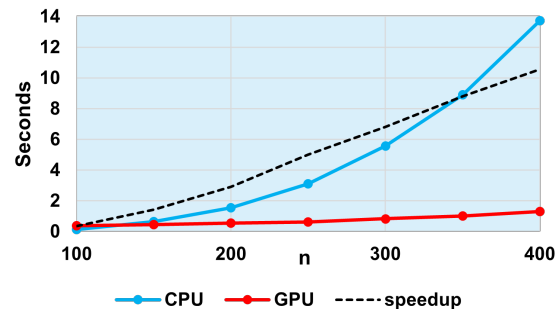


Figure 64: Total times (setup plus solve times) on 2 nodes of Spock using AMG-PCG with aggressive coarsening and multi-pass interpolation for a 3D diffusion problem with a 27-point stencil on a $n \times n \times n$ grid. The CPU runs were performed with 8 MPI tasks with 16 OpenMP threads per MPI task, the GPU runs were performed on 8 AMD MI100 GPUs.

Early Access System Experiences We have tested hypre performance on both Spock, see Figure 64, and Crusher. Figure 65 shows the Setup and Solve times for three different settings of AMG for increasing n of a 3D diffusion problem on a $n \times n \times n$ grid. Both CPU and GPU runs use the same settings. Solid lines use the default AMG settings, whereas dashed and dotted lines use one level of aggressive coarsening with two-stage and multipass interpolation, respectively. While generally systems in application problems would be in the range of up to $n = 300$, we here continue to increase the problem size until the GPU runs reach the memory limit to demonstrate memory usage for these different AMG settings. This is reached first for the default version, which requires a lot of memory, but also achieves the lowest number of iterations, followed by the use of aggressive coarsening with two-stage interpolation. The use of one level of aggressive coarsening combined with multipass interpolation allows solving the largest system illustrated using GPUs, since it has lower memory requirements at the cost of having the worst convergence.

hypre release v2.24.0 contains support for Intel GPUs for the structured interface and solvers via SYCL and Kokkos and for the solve phase of unstructured solvers via SYCL. All these features have been tested on Arcticus.

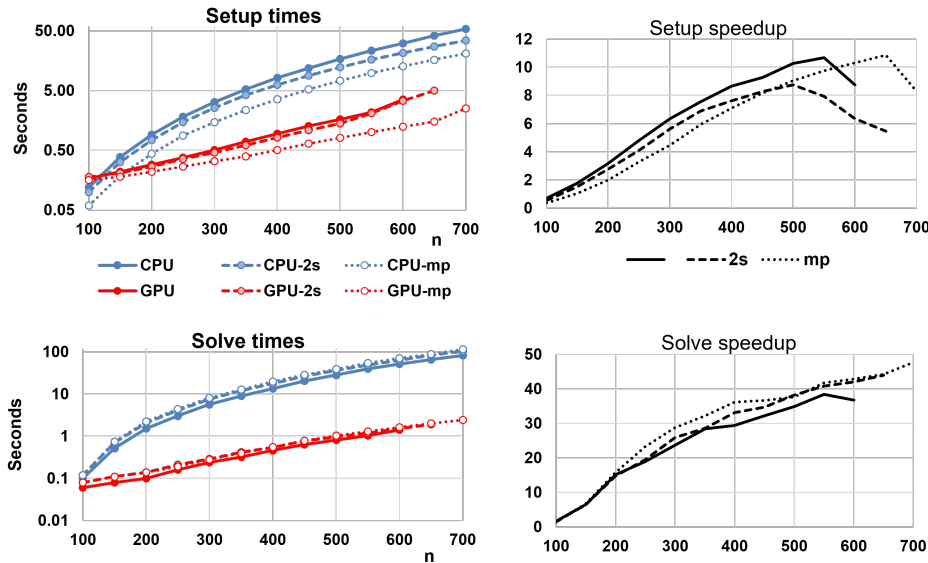


Figure 65: Setup, solve times and Speedups of GPU over CPU performance on 1 node of Crusher using AMG-PCG with three different settings for a 3D diffusion problem with a 27-point stencil on a $n \times n \times n$ grid. The CPU runs were performed with 64 MPI tasks, the GPU runs were performed on 8 AMD MI250 GPUs. Solid lines use the GPU default settings, whereas dashed and dotted lines use in addition one level of aggressive coarsening with two-stage and multipass interpolation, respectively.

Next Steps We will continue to add new GPU capabilities to hypr and improve the performance of current capabilities. We will thoroughly investigate the performance on AMD GPUs and begin porting to Intel GPUs. We will also test and evaluate performance of important hypr solvers on AMD and Intel GPUs, improve their performance, and port additional components if necessary.

Additionally, we will work with ECP application teams who are using hypr (e.g., ExaWind, ExaAM, and AMReX) to achieve the best performance by tuning the solvers for them and potentially implementing suitable algorithmic changes.

4.3.14 WBS 2.3.3.13 CLOVER

Mathematical libraries are powerful tools to make better use of Exascale architectural features and are central for application projects to efficiently exploit the available computing power. The high-level objective of CLOVER is to provide scalable, portable numerical algorithms that facilitate efficient application simulations on Exascale computers. With the intention of generating synergies by facilitating vivid cooperation among the distinct project focus efforts and expert knowledge transfer, CLOVER was designed as a merger of the heFFTe, SLATE, and Ginkgo projects, each being complementary in focus but similar in the need for hardware-specific algorithm design expertise: SLATE focuses on Exascale-capable dense linear algebra functionality; heFFTe's scope is providing robust and fast calculation for 2D and 3D FFT routines; Ginkgo delivers production-ready, preconditioned iterative solvers for GPU-accelerated systems. Together, these projects form a robust ecosystem of numerical base functionality for Exascale computers.

4.3.15 WBS 2.3.3.13 CLOVER Sub-project: heFFTe

Overview The Highly Efficient FFTs for Exascale (heFFTe) project provides sustainable high-performance multidimensional Fast Fourier Transforms (FFTs) for Exascale platforms [172, 173]. HeFFTe leverages established but ad hoc software tools that have traditionally been part of application codes, but not extracted as independent, supported libraries. The main objective of the heFFTe project is to: (1) Collect existing FFT capabilities from ECP application teams; (2) Assess gaps, extend, and make available various FFT capabilities as a sustainable math library; (3) Explore opportunities to build multidimensional FFTs while leveraging on-node concurrency from batched FFT formulations; (4) Focus on capabilities for Exascale platforms.

FFTs are used in many applications including molecular dynamics, spectrum estimation, fast convolution and correlation, signal modulation and many wireless multimedia applications. The distributed 3D FFT is one of the most important routines used in molecular dynamics (MD) computations, and its performance can affect MD scalability. The performance of the first principles calculations strongly depends on the performance of the FFT solver that performs many FFTs of size $\approx 10^7$ points in a calculation that we call batched FFT. Moreover, Poisson PDE-type equations arising from many engineering areas, such as plasma simulation and density fields, need to solve FFTs of size larger than 10^9 . More than a dozen ECP applications use FFT in their codes. ECP applications that require FFT-based solvers suffer from the lack of fast and scalable 3D FFT routines for distributed-heterogeneous parallel systems as the ones projected for the upcoming exascale computing systems. To address these needs, heFFTe functionalities are first delivered to CoPA projects using LAMMPS (molecular dynamics) and HACC (Hardware Accelerated Cosmology Code).

The heFFTe software stack is illustrated in the left-hand side of Figure 66, while the main components of the heFFTe framework are illustrated in the right-hand side of Figure 66.

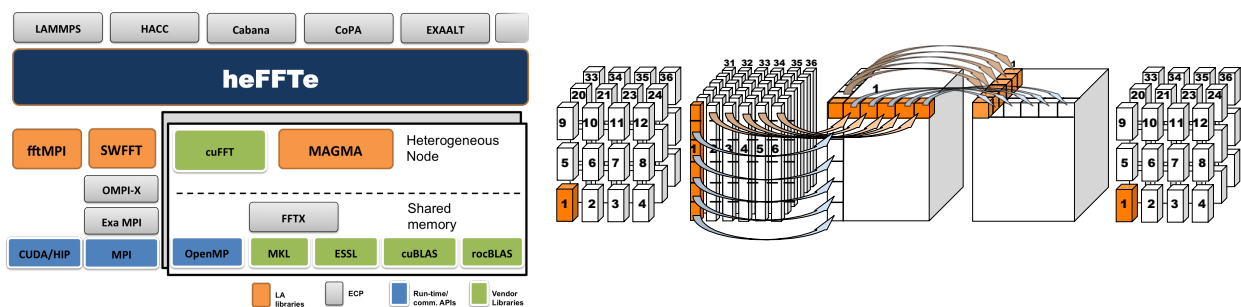


Figure 66: **Left:** heFFTe software stack. **Right:** 3D FFT computational pipeline in heFFTe with (1) flexible API for application-specific input and output, including bricks/pencils; (2) efficient packing/unpacking and MPI communication routines; and (3) efficient 1D/2D/3D FFTs on the node.

Key Challenges Each element of the software stack described above faces key challenges to successful implementation and deployment.

Communication Costs Communication costs are main bottleneck on current systems; this includes low node bandwidth (relative to high compute capabilities) and suboptimal accelerator-aware MPI communications that can lead to some strong scalability issues [174].

Application Specifics ECP applications that require FFT-based solvers suffer from the lack of fast and scalable FFTs for distributed-heterogeneous parallel systems as the ones projected for the upcoming exascale computing systems. Also, ECP applications need different application-specific versions of FFTs, and dictate parallelism and data distributions (where is the data, how is distributed, what is the parallelism, etc.). This requires application knowledge and API designs with a suitable modular high-performance implementation that is flexible and easy to use and integrate in ECP applications.

Performance Portability Performance portability across different architectures is always a challenge. This is further exacerbated due to the many application and hardware-specific FFT versions needed.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each challenge area.

Communications and GPU Optimizations FFTs are communication bound and a main focus in heFFTe is on algorithmic design to minimize communication and efficient GPU implementations [175, 176, 174, 177]. Other strategies include the use of mixed-precision calculations [178, 179] and data compression for reduced communications (including lossy, e.g., using ZFP compression) [152].

Evolving Design heFFTe is designed to support the fftMPI and SWFFT functionalities, which are already integrated in ECP applications. Thus, heFFTe benefits directly these applications and provides integrated solutions. More functionalities and application-specific optimizations will be added through heFFTe backends to support various ECP applications.

Autotuning Performance portability will be addressed through use of standards (like 1D FFTs from vendors), portable linear algebra (LA) using MAGMA [180], and parameterized versions that will be autotuned across architectures [181, 182].

Recent Progress The heFFTe team completed two main milestones involving software releases adding numerous stability, performance, and scalability enhancements, as well as new functionalities [174]. HeFFTe 2.1 was released in April 2021, and heFFTe 2.2 was released in October 2021. HeFFTe 2.1 added support for multidimensional FFTs and optimizations for real data. This included the development of R2C and C2R FFTs and their integration in heFFTe and specific optimizations in ECP applications. Support and optimizations was extended for AMD GPUs, dependence on MAGMA was added, as well as spack installation, and integration of heFFTe in xSDK. HeFFTe was also integrated in CoPA projects and ExaAM/Meumapps with new application-specific optimizations, tuning, and added Intel GPU support. HeFFTe 2.2 concentrated on adding support and optimization of the HIP and Intel GPU backends to heFFTe. HeFFTe’s functional and performance portability on these architectures and multicore CPUs was established [177]. Multidimensional FFTs for fast discrete convolution, cosine (DCT), and sine (DST) transforms were also added with support for Nvidia, AMD, and Intel GPUs. HeFFTe has demonstrate very good strong scalability and performance that is close to 90% of the roofline peak on the newly added in heFFTe v2.2 R2C, C2R, convolution, DCS and DST transformations [174, 173] (see Figure 67).

An FFT benchmark for a number of FFT libraries, including a design study and evaluation of FFT codes used in the ECP applications, was also developed [183].

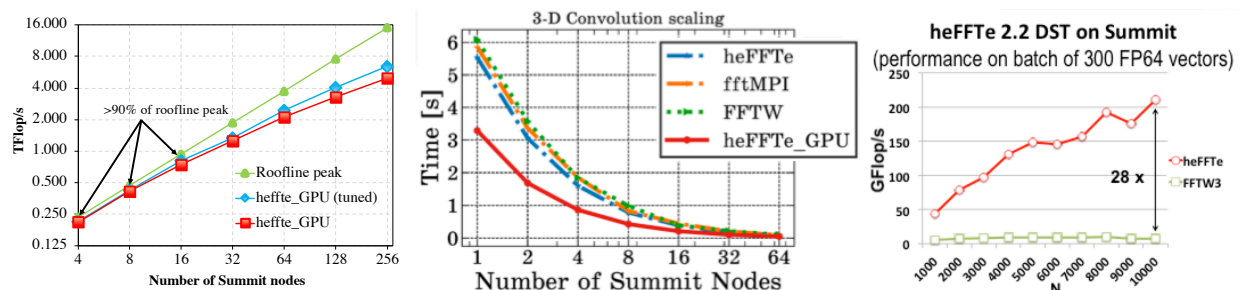


Figure 67: **Left:** heFFTe strong scalability on 1024³ FFT on up to 256 nodes (6× V100 GPUs; double complex arithmetic; starting and ending with bricks; performance assumes $5N^3 \log_2 N^3$ flops). **Middle:** Convolution of 512³ multidimensional arrays in double complex arithmetic. **Right:** Performance of the Discrete Sine Transformation (DST) in heFFTe 2.2.

Preliminary Experiences on Early Access systems heFFTe is building and passing tests and benchmarks on all Early Access systems (EAS). Current focus in particular is on improving the performance on the Spock and Crusher EAS that feature AMD GPUs. The benefits of the latest addition to heFFTe – batched multidimensional FFT functionalities – are illustrated in Figure 68 running on Spock for application sizes as needed in the NwChemEx ECP project.

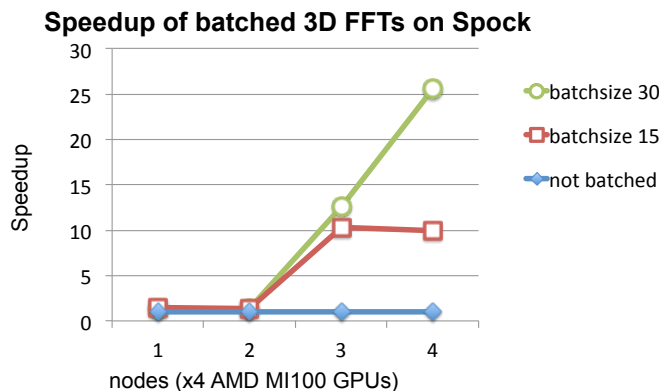


Figure 68: Speedup of batching FFTs using heFFTe on Spock. Shown is strong scalability of batched vs. not batched FFTs of size 32^3 , which are realistic application sizes as needed in NWChemEx.

Next Steps Next steps of work are adding mixed-precision algorithms and approximate FFTs that allow trade-off of speed vs. accuracy. Autotuning framework that hides backend selection and other parameters from users will be added, as well as improved GPU-aware MPI Alltoally routines. Existing FFT libraries will be evaluated through the FFT benchmark [183].

4.3.16 WBS 2.3.3.13 CLOVER Sub-project: SLATE

Overview SLATE (Software for Linear Algebra Targeting Exascale) provides fundamental dense linear algebra capabilities to the DOE and the HPC community at large. To this end, SLATE provides parallel BLAS (basic linear algebra subprograms), norms, linear system, least squares, singular value, and eigenvalue solvers. The ultimate objective of SLATE is to replace the venerable ScaLAPACK (Scalable Linear Algebra PACKage) library, which has become the industry standard for dense linear algebra operations in distributed-memory environments, but is past its end of life and can't be readily retrofitted to support GPUs.

Primarily, SLATE aims to extract the full performance potential and maximum scalability from modern multi-node HPC machines with many cores and multiple GPUs per node. This is accomplished in a portable manner by relying on standards such as MPI and OpenMP. Figure 69 shows the role of SLATE in the ECP software stack.

SLATE also seeks to deliver dense linear algebra capabilities beyond the capabilities of ScaLAPACK, including new features such as communication-avoiding and randomized algorithms, as well as the potential to support variable size tiles and block low-rank compressed tiles.

Key Challenges Each element described above faces key challenges to successful implementation and deployment.

Facing Harsh Hardware Realities SLATE targets a difficult hardware environment, where virtually all the processing power is on the GPU side. Achieving efficiency requires aggressive offload to GPU accelerators and optimization of communication bottlenecks.

Facing Harsh Software Realities SLATE uses cutting-edge software technologies, including modern C++ features and recent extensions to the OpenMP standard, which may not be fully supported by compilers and their runtime environments. Standardized solutions for GPU acceleration are still in flux.



Figure 69: SLATE in the ECP software stack.

Solution Strategy To mitigate these challenges, the team proposes the following actions.

Evolving Design Due to the inherent challenges of designing a software package from the ground up, the SLATE project started with a careful analysis of the existing and emerging implementation technologies [184], followed by an initial design [185] that has solidified [186], but we continue to reevaluate and refactor as needed.

Focus on GPUs Efficient GPU acceleration is the primary focus of performance engineering efforts in SLATE. Where applicable, highly optimized vendor implementations of GPU operations are used, such as the batched `gemm` routine. Where necessary, custom GPU kernels are developed, as for computing matrix norms. Care is taken to hide communication by overlapping it with GPU computations.

Community Engagement The SLATE team interacts on a regular basis with the OpenMP community, represented in ECP by the SOLLVE project, and with the MPI community, represented in ECP by the OMPI-X project and the Exascale MPI project. The SLATE team also engages the vendor community through our contacts at HPE Cray, IBM, Intel, NVIDIA, AMD, and ARM.

Recent Progress Recent progress in this area includes several key developments.

Port to AMD and Intel Platforms Originally, SLATE was developed using NVIDIA CUDA. We ported BLAS++ to the AMD ROCm and Intel oneAPI platforms, and use it as a portability layer for SLATE. Our CUDA kernels were ported to HIP using AMD’s `hipify` tool. We are also porting kernels to OpenMP Offload, which provides an option for all platforms (NVIDIA, AMD, and Intel GPUs).

Performance Improvements We improved performance of several major routines, including Cholesky, QR, eigenvalue, and singular value decompositions. More components of Cholesky and QR are now on the GPU. For the eigenvalue problem, we parallelized a major component and accelerated it using GPUs, resulting in significant improvements, as shown in Figure 70.

Added Eigenvector Computation We added computation of eigenvectors to the eigenvalue solver, and are working on the divide-and-conquer algorithm, which exhibits better performance than the QR iteration algorithm for computing eigenvectors.

Developments are documented in SLATE Working Notes.¹⁴

Preliminary Experiences on Early Access systems The SLATE team has been using the Early Access Systems to port SLATE to the AMD ROCm and Intel oneAPI GPU architectures. This includes machines with pre-release hardware and software, as well as the recent Spock and Crusher machines with AMD CPU and GPU hardware at Oak Ridge National Laboratory. We have used these engagements to benchmark early versions of the system software stack, and reported various performance and correctness issues to AMD, Intel, and Cray. Early results on Spock in Figure 71a show the SLATE matrix multiply (`dgemm`) achieves good

¹⁴<http://www.icl.utk.edu/publications/series/swans>

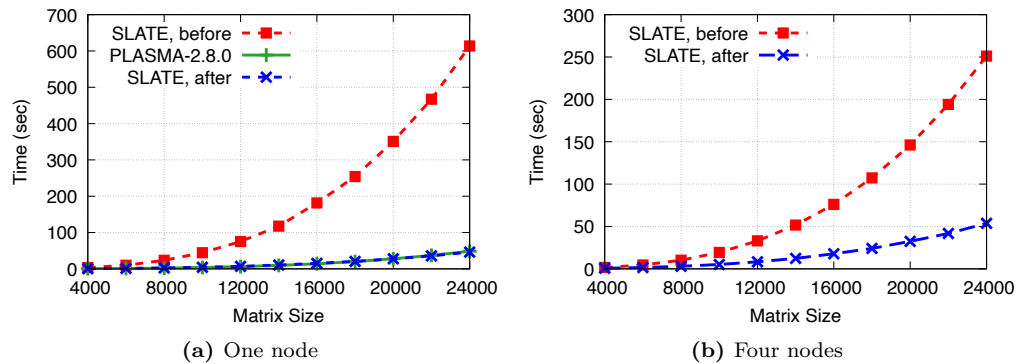


Figure 70: Performance improvement in Hermitian reduction to band for eigenvalue problem. Each node has 2×10 core Intel Haswell E5-2650 v3.

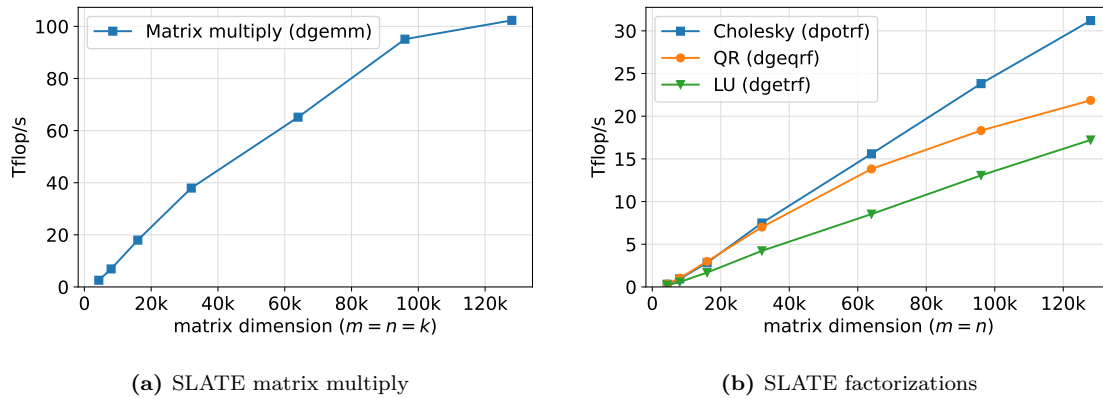


Figure 71: Performance on Spock (4 nodes, 16 AMD MI100 GPUs).

performance using multiple nodes and multiple GPUs. We have also demonstrated the Cholesky, LU, and QR factorizations (`dpotrf`, `dgeqrf`, `dgetrf`) in Figure 71b, and are continuing work to tune the performance, particularly of communication since AMD has made direct GPU-to-GPU MPI communication efficient.

Next Steps The following efforts have been identified for the next phase of the project.

CA-QR and CA-LU Communication avoiding (CA) routines reorganize the traditional computation to minimize the number and volume of communications. Since communication is a major bottleneck, both between nodes and between different memories within a node, we expect CA algorithms to improve SLATE’s performance. In particular, we are optimizing QR factorizations with a focus on tall-skinny matrices, such as solving $10,000,000 \times 1000$ least squares problems. There are several avenues to explore including CA-QR and Cholesky-QR algorithms.

Optimize Level 2 BLAS For parallel BLAS, the initial implementations focused on the case where the output matrix C was large. However, the output matrix is often a single vector, corresponding to Level 2 BLAS, or a few vectors. For instance, solving a single right-hand side b in $Ax = b$. These operations require a different algorithm, so that the computation occurs where the large input matrix A resides, rather than where the output matrix b or x resides. This lack of Level 2 BLAS optimizations was identified as a major bottleneck in our mixed-precision iterative refinement algorithms. We will optimize our existing matrix-multiplication routines to handle these cases, automatically switching algorithms as needed.

4.3.17 WBS 2.3.3.13 CLOVER Sub-project: Ginkgo

Overview Ginkgo¹⁵ is a modern linear algebra library engineered towards performance portability, and productivity. To achieve these goals, the library design is guided by combining ecosystem extensibility with heavy, architecture-specific kernel optimization using the platform-native languages CUDA (NVIDIA GPUs), HIP (AMD GPUs), DPC++ (Intel GPUs) and OpenMP (Intel/AMD/ARM multicore). Ginkgo is part of the extreme-scale Software development Kit (xSDK), part of the Extreme-Scale Scientific Software Stack (E4S), and has already been integrated as a backend into simulation libraries like deal.II, MFEM, and HyTeG.

Key Challenges The extreme levels of hardware concurrency available in the GPU-accelerated nodes must be reflected in fine-grain parallelism in the numerical building blocks. To that end, an increasing variety of hardware designs and hardware-native programming languages requires a library design that enables platform portability without sacrificing performance.

Applications that build upon the fast solution of many independent moderate-sized sparse linear systems require batched sparse linear algebra functionality, and applications that build upon matrix-free methods require the flexibility to compose linear solvers out of library-native and customized external functionality. Moreover, the arithmetic performance of processors is growing much faster than the memory bandwidth and interconnect speed, and this requires exploring innovative strategies to reduce the pressure on all cache/memory levels.

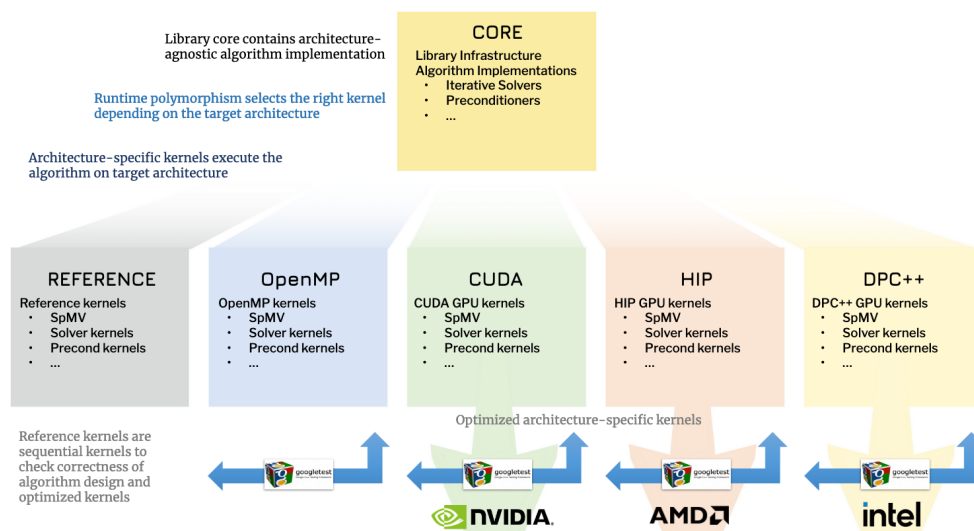


Figure 72: The portability design of the Ginkgo math library enables high performance kernel implementations in the vendor-supported programming language.

Solution Strategy The Ginkgo team is addressing these challenges by striving for platform portability, modularity, extensibility, and hardware-aware algorithm design.

Architecture-Portable Software Design Ginkgo [187] employs a design that decouples the algorithm implementations from the hardware-specific kernel implementations, thereby acknowledging the importance of platform portability and allowing for architecture-specific kernel optimization in the vendor language, see Figure 72.

Modularity, Flexibility, and Extensibility Ginkgo employs a linear operator abstraction for all functionality, which allows for flexibility in combining functionality and interfacing external operators.

¹⁵<https://github.com/ginkgo-project/ginkgo>

Sustainability Efforts Ginkgo adheres the Better Scientific Software (BSSw) design principles [188] that ensure production-quality code by featuring unit testing, automated configuration and installation, Doxygen code documentation, as well as a continuous integration and continuous benchmarking framework [189]. Ginkgo is an open source effort licensed under the BSD 3-clause and included in the xSDK and E4S software packages.

Fine-Grain Parallelism Ginkgo features linear algebra building blocks and advanced algorithms for preconditioning and solving linear systems that can efficiently leverage the concurrency of modern GPUs, including (incomplete) parallel factorizations, parallel sparse triangular solves, parallel matrix operations, and parallel iterative methods.

Mixed-Precision Methods Ginkgo features a memory accessor that encapsulates on-the-fly compression for decoupling the memory precision from the arithmetic precision. This allows accelerating memory-bound algorithms that can compensate or tolerate some information loss in the memory operations. The memory accessor can also be used to increase the result accuracy of memory-bound kernels that benefit from using a more complex precision format in the arithmetic operations without performance loss.

Batched Preconditioned Iterative Solvers Ginkgo contains high performance batched iterative solvers that handle the concurrent solution of a set of independent sparse linear systems of moderate size. The batched iterative solvers allow for some flexibility in terms of the preconditioner and monitor the system-individual convergence without performance degradation.

Recent Progress First, the Ginkgo library realized native support for Intel GPUs (via DPC++) [190]. Also, MFEM treating Ginkgo as a numerical backend allows accelerating MFEM simulations with AMD GPUs, Intel GPUs, and NVIDIA GPUs. The Ginkgo library has also been expanded with new advanced preconditioners such as ISAI and Multigrid, also supporting mixed-precision capabilities [191]. Based on the memory accessor, the Ginkgo team deployed a Compressed Basis GMRES (CB-GMRES) solver that outperforms the standard GMRES solver by storing the Krylov basis vectors in lower precision, therewith accelerating the memory access [192]. Finally, the Ginkgo team successfully employed the batched iterative solvers for the hydrodynamic problems arising in the PeleLM simulations and the gyrokinetic problems arising in the XGC simulations [193].

Next Steps The following efforts have been identified for the next phase of the project.

Deployment of Multinode Functionality Ginkgo already has experimental support for multi-node execution via an executor-agnostic MPI layer. We will extend the support and make it production-ready.

Deployment of GPU-resident Sparse Direct Solvers In response to an urgent need of national power grid simulations, we will develop GPU-resident sparse direct solvers.

Block-Versions of the Parallel Incomplete Factorization Preconditioner To better reflect the properties of the ECP application projects, we will deploy blocked versions of the ParILU and ParILUT parallel ILU and parallel threshold ILU preconditioners in the Ginkgo software library.

Problem-Specific Preconditioners for MFEM In collaboration with the ECP CEED cluster, we will design problem-specific mixed precision preconditioners for matrix-free finite element simulations.

Early Access Systems (EAS) Experiences Due to its backend model, Ginkgo has successfully been extended to HIP, with support fully released in version 1.2.0 in July 2020 and DPC++ released in version 1.4.0 in August 2021. The Ginkgo team has been able to continue the development of these backends, work on new experimental features (e.g., batched and distributed), and improve existing functionality performance when needed, in particular thanks to EAS availability with the Perlmutter, JLSE Arcticus, Spock and Crusher machines. In the following, we report results obtained on the EAS Crusher for some of the single node Ginkgo functionality.

On the EAS Crusher system, we evaluate the performance of some of Ginkgo’s iterative solvers and their essential building block, the Sparse Matrix-Vector product operation (SpMV), on one GCD of a MI250X.¹⁶ In these figures, each dot is a matrix from the SuiteSparse Matrix Collection¹⁷. In Figure 73 we show the bandwidth of the Ginkgo SpMV operation with the CSR and ELL formats. For matrices with a large nonzero count, both SpMV formats reach up to 87% of the 1.6 TB/s peak bandwidth for one GCD. In Figure 74 we benchmark the bandwidth of the CG, and GMRES solvers. For each matrix and solver combination, Ginkgo’s benchmarking setup first runs the SpMV benchmark, selects the fastest matrix format for the matrix in question and runs the solver benchmark. This solver benchmark consists of 10000 unpreconditioned iterations for each solver type and all results are normalized over a single iteration. We see that the CG solver reaches a bandwidth of up to 80% of the theoretical peak, while on average GMRES can have a slightly lower efficiency and be more affected by the sparsity pattern of the matrix.

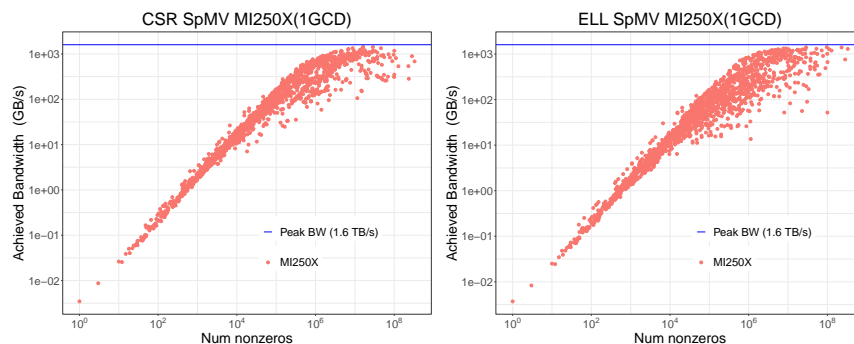


Figure 73: Achieved bandwidth (GB/s) of the Ginkgo CSR (left), and ELL (right) SpMV operations on one GCD of the Crusher MI250X GPUs.

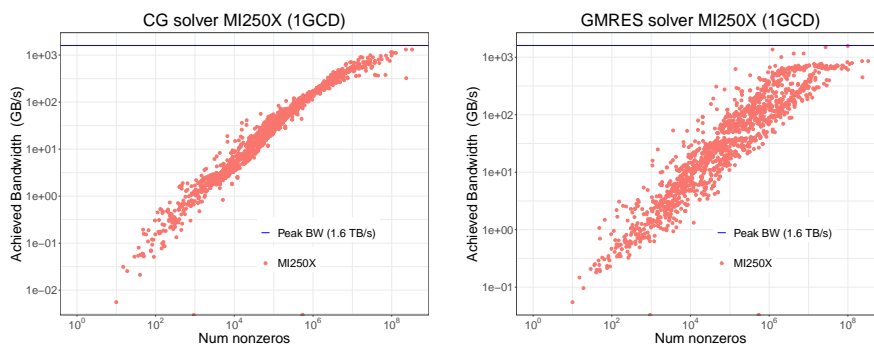


Figure 74: Achieved Bandwidth (GB/s) of the Ginkgo CG (left) and GMRES (right) solvers on one GCD of the Crusher MI250X GPUs.

4.3.18 WBS 2.3.3.14 ALExa

Overview The ALExa project (Accelerated Libraries for Exascale) focuses on preparing the ArborX, DataTransferKit (DTK), Tasmanian, and ForTrilinos libraries for exascale platforms and integrating these libraries into ECP applications. These libraries deliver capabilities identified as needs of ECP applications: (1) the ability to execute performance portable spatial searches between arbitrary sets of distributed geometric objects (ArborX); (2) the ability to transfer computed solutions between grids with differing layouts on parallel accelerated architectures, enabling multiphysics projects to seamlessly combine results from different computational grids to perform their required simulations (DTK); and (3) the ability to construct fast

¹⁶For platform details, we refer to [ORNL Crusher Documentation](#) and [AMD CDNA-2 Architecture Whitepaper](#).

¹⁷<https://sparse.tamu.edu/>

and memory efficient surrogates to large-scale engineering models with multiple inputs and many outputs, enabling uncertainty quantification (both forward and inverse) as well as optimization and efficient multiphysics simulations in projects such as ExaStar (Tasmanian); and (4) the ability to automatically interface Fortran-based codes to existing large and complex C/C++ software libraries, such as Trilinos advanced solvers that can utilize next-generation platforms.

These capabilities are being developed through ongoing interactions with our ECP application project collaborators to ensure they will satisfy requirements of these customers. The libraries in turn take advantage of other ECP/SW capabilities currently in development, including Trilinos, Kokkos, and SLATE. The final outcome of the ECP project will be a set of libraries deployed to facilities and also made broadly available as part of the xSDK4ECP project.

ArborX Purpose: ArborX is an open-source library designed to provide performance portable algorithms for geometric search.

Significance: General geometric search capabilities are needed in a wide variety of applications, including the generation of neighbor lists in particle-based applications (e.g., molecular dynamics or general N-body dynamics simulations), density-based clustering analysis (e.g., halo finding or DBSCAN in cosmology) and mesh-mesh interactions such as contact in computational mechanics and solution transfer in multiphysics simulations.

Capabilities: Performance-portable search capabilities include shared-memory and GPU implementations of spatial tree construction; shared memory and GPU implementations of various spatial tree queries; MPI front-end for coordinating distributed spatial searches between sets of geometric objects with different decompositions; communication plan generation based on spatial search results; density-based clustering algorithms (DBSCAN).¹⁸

DTK Purpose: Transfers computed solutions between grids with differing layouts on parallel accelerated architectures.

Significance: Coupled applications frequently have different grids with different parallel distributions; DTK is able to transfer solution values between these grids efficiently and accurately.

Capabilities: Mesh and mesh-free interpolation capabilities include multivariate data interpolation between point clouds and grids; compactly supported radial basis functions; nearest-neighbor and moving least square implementations; support for standard finite-element shape functions and user-defined interpolants; common applications include conjugate heat transfer, fluid structure interaction, and mesh deformation.¹⁹

Tasmanian (Toolkit for Adaptive Stochastic Modeling and Non-Intrusive Approximation) Purpose: Constructs efficient surrogate models for high-dimensional problems and performs parameter calibration and optimization geared towards applications in uncertainty quantification (UQ).

Significance: UQ pertains to the statistical properties of the output from a complex model with respect to variability in multiple model inputs; large number of simulations are required to compute reliable statistics which is prohibitive when dealing with computationally expensive engineering models. A surrogate model is constructed from a moderate set of simulations using carefully chosen input values; analysis can then be performed on the efficient surrogate.

Capabilities: Sparse grids capabilities include surrogate modeling and design of experiments (adaptive multi-dimensional interpolation); reduced (lossy) representation of tabulated scientific data; high dimensional numerical quadrature; data mining and manifold learning.

DiffeRential Evolution Adaptive Metropolis (DREAM) capabilities include Bayesian inference; parameter estimation/calibration; model validation. global optimization and optimization under uncertainty.²⁰

ForTrilinos (Fortran Trilinos) Purpose: ForTrilinos provides a seamless pathway for large and complex Fortran-based codes to access Trilinos without C/C++ interface code. This access includes Fortran versions of Kokkos abstractions for code execution and data management. To provide this functionality, this project developed a Fortran-targeted extension to the SWIG (Simplified Wrapper and Interface Generator) tool.

¹⁸<https://github.com/ArborX/ArborX>

¹⁹<https://github.com/ORNL-CEES/DataTransferKit>

²⁰<http://tasmanian.ornl.gov>

Applied to Trilinos, it generates object-oriented Fortran 2003 interface code that closely mirrors the Trilinos C++ API.

Significance: The ECP requires the successful transformation and porting of many Fortran application codes in preparation for ECP platforms. A significant number of these codes rely upon the scalable solution of linear and nonlinear equations. The Trilinos Project contains a large and growing collection of solver capabilities that can utilize next-generation platforms, in particular scalable multicore, manycore, accelerator and heterogeneous systems. Since Trilinos is written primarily in C++, its capabilities are not available to other programming languages. ForTrilinos bridges the gap between the needs of Fortran app developers and the capabilities of Trilinos. Furthermore, the technology used to generate the Fortran–C++ bindings in ForTrilinos is capable of exposing any number of C++ libraries to Fortran exascale app developers.

SWIG capabilities: ForTrilinos provides an inversion of control functionality that enables custom extensions of the Trilinos solvers implemented in downstream Fortran apps. Although this capability is not yet comprehensive, the goal of this project is to provide functional and extensible access Trilinos on next-generation computing systems. Several examples of ForTrilinos are being demonstrated within Fortran-based ECP codes to help them meet simulation goals and illustrate the technology to other Fortran-based ECP codes. Additionally, the SWIG technology underpinning ForTrilinos is being applied to other C++-based ECP ST subprojects to expose their capabilities to Fortran apps.²¹

Key Challenges Each element described above faces key challenges to successful implementation and deployment.

ArborX Search procedures to locate neighboring points, mesh cells, or other geometric objects require tree search methods difficult to optimize on modern accelerated architectures due to vector lane or thread divergence. A flexible interface for calling user kernels on a positive match as well as modifying traversal algorithms in a task-specific manner are crucial to achieving the best performance.

DTK General data transfer between grids of unrelated applications requires many-to-many communication which is increasingly challenging as communication to computation ratios are decreasing on successive HPC systems. Maintaining high accuracy for the transfer requires careful attention to the mathematical properties of the interpolation methods and is highly application-specific.

Tasmanian Extracting statistical information from a Tasmanian surrogate (or using the surrogate in a multi-physics simulation) requires the collection of a large number of samples, which is not feasible without GPU acceleration. The GPU accelerated surrogate evaluations require both custom kernels corresponding to the different types of basis functions as well as both sparse and dense linear algebra methods (BLAS level 2 and 3). Porting the capabilities and optimizing the performance across different divergent architectures is challenging.

ForTrilinos Developing the interfaces to the C++ libraries that provide access to cutting-edge research, such as Trilinos, is of significant benefit to Fortran community. However, such interfaces must be well documented, sustainable and extensible, which would require significant amount of resources and investment. This is further complicated by the requirements to support heterogeneous platforms (e.g., GPUs) and inversion-of-control functionality. The manual approach to such interfaces has been shown to be unsustainable as it requires interface developers to have in-depth expertise in multiple languages and the peculiarities in their interaction on top of the time commitment to update the interfaces with changes in the library.

ForTrilinos addresses both the issue of reducing interface generation cost through investment in tool configuration and usage to make the process as automatic as possible, and the issue of providing the full-featured interface to Trilinos library, including access to manycore, accelerator and heterogeneous solver capabilities in Trilinos.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each effort.

²¹<https://github.com/trilinos/ForTrilinos>

ArborX ArborX builds on a MPI+Kokkos programming model to deploy to all DOE HPC architectures. Extensive performance engineering has yielded implementations that are both as performant in serial as state-of-the-art libraries while also expanding on the capability provided by other libraries by demonstrating thread scalability on both GPU and multi-core CPU architectures. Working with both synthetic as well as real data from applications (e.g., HACC) ensures wide performance testing coverage.

DTK State-of-the-art, mathematically rigorous methods are used in DTK to preserve accuracy of interpolated solutions. Algorithms are implemented in a C++ code base with extensive unit testing on multiple platforms. Trilinos packages are used to support interpolation methods. Kokkos is used to achieve performance portability across accelerated platforms.

Tasmanian The C++ CUDA/HIPO kernels within Tasmanian are templated exposing numerous performance tweaks and tuning parameters that can be adjusted to perform well on a corresponding Nvidia/AMD system. The kernels are also ported to DPC++/SYCL to allow for the utilization of Intel GPUs. Tasmanian also links to general GPU-BLAS interfaces that can utilize any of the accelerated backends, e.g., cuBlas, rocBlas, MKL and MAGMA.

ForTrilinos ForTrilinos defines several SWIG-Fortran modules that generate Fortran-2003 interfaces to C++ Trilinos solver classes. ForTrilinos provides a high-level interface for applications to access nonlinear and eigenvalue solvers in addition to low-level Trilinos classes. The SWIG-Fortran methodology is applicable to other ECP projects and has been used strategically to provide low-maintenance Fortran wrappers for several other ST codes.

Recent Progress Recent progress includes developments in ArborX, Tasmanian, and ForTrilinos.

ArborX The team developed several new features in ArborX to support applications. For the analysis of the cosmological data in the ExaSky project, we developed a fast performance-portable algorithm to compute DBSCAN (Density-based spatial clustering of applications with noise) clustering. It speeds up the halo-finding algorithm (one of the most expensive elements in the cosmological analysis pipeline) by more than 15x. For the stretched inclined geometries of the wind turbine simulations in the ExaWind project, ArborX now provides a new k-DOP bounding volume.

Tasmanian Work with partner application ExaStar (2.2.3.01) performed multiphysics simulations on OLCF Summit using the Nvidia Volta GPUs. The Fortran interface of Tasmanian was extended (using Swig-Fortran) to expose the necessary C++ methods. Utilizing the GPUs yields a near 10x performance boost, which is consistent with the Tasmanian stand-alone test indicating that the Thornado-Tasmanian interface does not incur appreciable overhead.

ForTrilinos This year has focused on wrapping up development of ForTrilinos and applying SWIG-Fortran to other ECP projects. We released Flibcpp 1.0.0, a set of Fortran interfaces to the C++ standard library to supplement ForTrilinos. We released ForTrilinos 2.0.0 based on ForTrilinos 13. New documentation for Flibcpp and SWIG-Fortran were published as ORNL technical memoranda. SWIG-Fortran gained support for C and C++ complex numbers. We applied SWIG-Fortran to other ECP projects and C/C++ libraries, generating Fortran wrappers for METIS/ParMETIS, STRUMPACK, and Tasmanian.

Next Steps The following efforts have been identified for the next phase of the project.

ArborX Continue improving performance of both the core and application-specific kernels and algorithms on all available accelerators. Expand search algorithms to accommodate multi-dimensional data.

DTK Continue performance engineering campaign and deploy in a variety of applications.

Tasmanian Tune the existing GPU kernels on the next generation GPU architectures from Nvidia, AMD and Intel. Extend the Tasmanian-libEnsemble integration to allow for advanced workflow management of libEnsemble (2.3.3.06) to be coupled with the asynchronous sampling methods within Tasmanian.

ForTrilinos Complete documentation for ForTrilinos native Fortran interface. Continue maintenance and support for ForTrilinos, SWIG-Fortran, and related Fortran interfaces for other ECP projects.

Preliminary Experiences on Early Access systems

ArborX ArborX is building, and all the tests are passing on various EAS systems including Perlmutter, Spock, Crusher, and Iris. We are currently focusing on improving the performance on Crusher.

DTK DTK is building, and all the tests are passing on Spock and Crusher.

Tasmanian is building and testing on all EAS systems, and tests are included in the Spack package. Currently, performance testing and improvements is done on Spock and Crusher.

4.3.19 WBS 2.3.3.15 Sake

Overview Modern simulation codes running on high performance computing (HPC) machines often rely heavily on multiple libraries to provide core capabilities such as meshing, mathematical algorithms, I/O services and more. This approach is highly productive as it allows domain experts to focus on their core technical contributions. The Sake project focuses on the design and development of performance portable mathematical libraries within the Trilinos project and the Kokkos ecosystem. Specifically, the team provides new implementations of linear algebra methods optimized for the architectures planned for the upcoming exascale systems while using interfaces already defined in Trilinos allowing applications a smooth transition toward exascale readiness.

Sake has two software products: Trilinos and KokkosKernels. It is organized into three subprojects, Trilinos, KokkosKernels, and PEEKS, where PEEKS and KokkosKernels were formerly part of CLOVER, whereas the Trilinos subproject is new in ECP.

4.3.20 WBS 2.3.3.15 Sake Sub-project: Kokkos Kernels

Overview The Kokkos Kernels ²² subproject primarily focuses on performance portable kernels for sparse/dense linear algebra, graphs, and machine learning, with emphasis on kernels that are key to the performance of several ECP applications. We work closely with ECP applications to identify the performance-critical kernels and develop portable algorithms for these kernels. The primary focus of this subproject is to support ECP application needs, develop new kernels needed with an emphasis towards software releases, tutorials, boot camps and user support. The Kokkos Kernels project also works closely with several vendors (AMD, ARM, HPE/Cray, Intel, and NVIDIA) as part of both the ECP collaborations and NNSA's Center for Excellence efforts. These collaborations will enable vendor solutions that are targeted towards ECP application needs.

Key Challenges There are several challenges in allowing ECP applications move to the hardware architectures announced in the next few years. We highlight the four primary challenges here:

1. The next three supercomputers that will be deployed will have three different accelerators from AMD, Intel and NVIDIA. While we have been expecting diversity of architectures, three different architectures in such a short time frame adds pressure on the portability solutions such as Kokkos Kernels to optimize and support the kernels on all the platforms.
2. The design of several ECP applications and a software stack that rely on a component-based approach results in an extremely high number of kernels launches on the accelerators, which results in the latency costs becoming the primary bottleneck in scaling the applications.
3. The change in the needs of applications from device-level kernels to smaller team-level kernels. Vendor libraries are not ready for such a drastic change in software design.
4. The reliance of ECP applications on certain kernels that do not port well to the accelerator architectures.

²²<https://github.com/kokkos/kokkos-kernels>

These challenges require a collaborative effort to explore new algorithmic choices, working with the vendors to incorporate ECP needs into their library plans, to develop portable kernels from scratch, and to deploy them in a robust software ecosystem. The Kokkos Kernels project will pursue all of these choices in an effort to address these challenges.

Solution Strategy To mitigate these challenges, the team proposes the following actions.

Codesign Portable Kernels with Vendors and Applications We rely on codesign of Kokkos Kernels implementations for specializations that are key to the performance of ECP applications. This requires tuning kernels even up to the problem sizes that are of interest to our users. Once we have developed a version, we provide these to all the vendors so their teams can optimize these kernels even further in vendor-provided math libraries.

Emphasis on Software Support and Usability The Kokkos Kernels project devotes a considerable amount of time working with ECP applications, integrating the kernels into application codes, tuning for application needs, and providing tutorials and user support. We invest in delivering a robust software ecosystem that serves the needs of diverse ECP applications on all platforms of interest.

Invest in Algorithmic Research to Reduce Latency Costs and New Accelerator-Focused Approaches To resolve latency cost issues, the Kokkos Kernels team is considering several solutions from computer science perspectives and also from algorithmic applied mathematics perspectives. For example, from a computer science perspective, we are focusing on the use of streams or other latency reducing techniques such as cuda graphs. From the applied mathematics perspective we are developing new algorithms such as cluster-based approaches for preconditioners, such as Gauss-Seidel preconditioners, to reduce the number of kernel launches.

Recent Progress

- New batched linear solver algorithms are being developed in support of ECP applications (Pele, XGC). Specifically, a preconditioned batched GMRES is now available and provides good performance on multiple GPU architectures of interest to ECP.
- Support for block sparse row matrices (BsrMatrix) is now available and associated matrix-vector operation is available. Additional performance improvements are expected for the BsrMatrix mat-vec.
- Further progress on the SYCL backend has been made and are available in the develop branch of our repository. These changes will be included in release 3.6.0 and will result in further performance portability for the libraries and applications we support (Trilinos, PETSc, ExaWind).
- Kokkos Kernels team is currently updating documentation and wiki pages to keep up with release 3.6.0 new features. Adjustments to tutorial are being made when necessary to ensure the new capabilities are advertised appropriately.

Preliminary Experiences on Early Access systems Kokkos Kernels engages all ECP facilities to ensure the library development is taking into account the specificities of all the architectures applications will encounter. Some of the recent developments include the development of new performance algorithms that leverage tensor core capabilities of Nvidia V100 on Summit and A100 on Perlmutter. New support for rocBLAS and rocSPARSE has been deployed and tuning for our Kernels on the HIP backend has been performed. These have been tested on EAS Spock and Crusher at OLCF. Finally regular meetings are held with Argonne and Intel staff to ensure that the library is deployed on JLSE Arcticus and provides adequate capabilities to users. These meetings have resulted in significant improvements for our SYCL backend support.

Next Steps The Kokkos Kernels team has identified the following efforts for the next phase of the project.

A Major Software Release Kokkos ecosystem 4.0 release will be available to the ECP applications in FY 2022. This includes several new kernels that are requested by ECP applications, performance improvements of kernels that are already being used by ECP applications and further support for new SYCL and OpenMP Target to enable applications to be tested on Intel architectures.

Develop New Sparse Batched and Block-Sparse Kernels The team continues to work on sparse batched linear algebra kernels and associated dense batched kernels. Additionally, more support for BsrMatrix is forthcoming with Matrix-Matrix multiplication, Jacobi and Gauss-Seidel algorithms.

Collaborate with Vendors and Facilities The Kokkos Kernels' team is continually working with vendors and facilities to improve the software environment on available supercomputers and early access systems.

4.3.21 WBS 2.3.3.15 Sake Sub-project: Trilinos/PEEKS

Overview Trilinos is a large and widely used toolkit for scientific computing, with many users both at DOE labs, in academia, and in industry. This project is focused on making Trilinos ready for exascale. One part is to port a core set of Trilinos packages to relevant architectures (including NVIDIA, AMD, and Intel GPUs). The other part is to design algorithms that work well on accelerators and at large scale (the focus of the PEEKS sub-project).

Key Challenges The Trilinos software library needs to be adapted to the new architectures. This is a large undertaking as there are many packages. Therefore, we focus on a core subset of Trilinos related to linear algebra and solvers. We are sensitive that Trilinos has a large user base and therefore need to minimize any user interface changes. Performance portability across a wide range of platforms is a key challenge.

Developing scalable iterative (Krylov) solvers for the US leadership supercomputers deployed in ECP, we acknowledge three major challenges coming from the hardware architecture:

1. Performance portability for Krylov solvers to perform well on different architectures with a single code base such that the applications relying on Trilinos for their linear solver needs can smoothly transition to the new ECP machines.
2. Fine-grained parallelism in a single node that has to be exploited efficiently by the iterative solver and the preconditioner.
3. Rising communication and synchronization cost as the computational power is growing much faster than memory power, resulting in increased pressure on the bandwidth of all cache/memory levels.

These challenges require the redesign of existing iterative solvers with respect to higher parallelism, a reduced number of communication and synchronization points, favoring computations over communication, and possibly adopting multiprecision algorithms for efficient hardware utilization.

Solution Strategy The primary thrusts of the Trilinos/PEEKS project designed to meet these challenges are detailed below.

Performance Portability We plan to rely heavily on the Kokkos and Kokkos Kernels libraries as that provide kernels that are performance portable across a variety of platforms, including CPU and GPU (NVIDIA, AMD, Intel). There is still much porting work, as some packages rely on UVM (Unified Virtual Memory), which is not widely supported. We will ensure the readiness of the following four solver packages on ECP platforms: distributed linear algebra (Tpetra), Krylov solvers (Belos), algebraic preconditioners and smoothers (Ifpack2), and direct solver interfaces (Amesos2).

Low-Synchronization Krylov Methods We will develop and deploy pipelined and communication-avoiding Krylov methods in production-quality code, and we are actively collaborating with the ECP ExaWind project to integrate our new features into their application. Another bottleneck we will address is the orthogonalization needed in GMRES (such as CGS and MGS).

Recent Progress Recent progress includes developments in UVM dependencies, HIP backend support, and low-synch orthogonalization.

Removal of Dependency on UVM We removed the dependence of the four solver packages on UVM. On NVIDIA GPUs, UVM simplifies coding by providing automatic data migration between the CPU and GPU memory. However, there are concerns about how well this will be supported, especially in term of performance, on future GPU systems such as Frontier and Aurora. Hence, removing UVM usage was a prerequisite for preparation to run on Frontier and Aurora platforms. More generally, the dependence on UVM posed a risk in term of the solver performance, while explicitly managing the data movement will likely improve solver performance.

HIP Backend Support We ensured the Trilinos solver stacks run on the Frontier early access system (Spock) using HIP backend (without UVM) and resolved initial performance problems. Several performance optimizations remain open which we plan to address this year.

Low-Synch Orthogonalization In collaboration with CU Denver and NREL, we developed and implemented a novel low-synch version of Classical Gram-Schmidt (CGS), called DCGS2. By delaying orthogonalization and norms, we reduce the synchronization requirement to once per iteration (even with two passes of CGS). The method is more numerically stable than previous attempts at low-synch orthogonalization. Preliminary results on Summit show that the new DCGS2 outperforms the current CGS2 and MGS methods, and achieves up to 66 Gflop/s on 192 GPUs. A version of this method is currently being integrated into Trilinos/Belos.

Preliminary Experiences on Early Access systems The team has been actively using the Frontier early access system (Spock) for initial solver performance tuning with the AMD GPUs. The early access has allowed us to identify the performance issues in some of the Trilinos computational kernels and resolve some of the issues (e.g., please see Kokkos-Kernels PR #1050, where the time to solution for solving a Nalu-Wind momentum problem was improved by 220× on Spock by tuning performance of BLAS kernels). We have also built our Trilinos solvers on Crusher and are now looking to further optimize the solver performance (e.g., using rocBLAS/rocSparse).

Next Steps The following efforts have been identified for the next phase of the project.

1. Support OpenMP and SYCL backends and perform baseline performance assessments to determine which kernels or algorithms need performance optimization on platforms that are relevant to ECP.
2. Conduct performance optimization using HIP, SYCL, and OpenMP backends on the platforms and problems that are relevant to ECP.
3. Implement graph assembly on a GPU to enable faster matrix assembly and to avoid data movement between host and device.
4. Design an unified solver interface to all the Trilinos solvers in order to ease the effort of ECP applications to switch between solvers and also to compose different solvers in one framework, both of which are needed for the success of ECP application projects especially when the solver options differ across architectures.
5. Deploy new orthogonalization method (DCGS2 or similar) in Trilinos/Belos, likely as an option in GMRES (which is used by several ECP applications).
6. Study Block Krylov methods, which may be particularly well suited to GPUs, and also appear to reduce communication.

4.4 WBS 2.3.4 DATA & VISUALIZATION

End State: A production-quality storage infrastructure necessary to manage, share, and facilitate analysis of data in support of mission critical codes. Data analytics and visualization software that effectively supports scientific discovery and understanding of data produced by Exascale platforms.

4.4.1 *Scope and Requirements*

Changes in the hardware architecture of Exascale supercomputers will render current approaches to data management, analysis and visualization obsolete, resulting in disruptive changes to the scientific workflow and rendering traditional checkpoint/restart methods infeasible. A major concern is that Exascale system concurrency is expected to grow by five or six orders of magnitude, yet system memory and input/output (I/O) bandwidth/persistent capacity are only expected to grow by one and two orders of magnitude, respectively. The reduced memory footprint per FLOP further complicates these problems, as does the move to a hierarchical memory structure. Scientific workflow currently depends on exporting simulation data off the supercomputer to persistent storage for post-hoc analysis.

On Exascale systems, the power cost of data movement and the worsening I/O bottleneck will make it necessary for most simulation data to be analyzed in situ, or on the supercomputer while the simulation is running. Furthermore, to meet power consumption and data bandwidth constraints, it will be necessary to sharply reduce the volume of data moved on the machine and especially the data that are exported to persistent storage. The combination of sharp data reduction and new analysis approaches heighten the importance of capturing data provenance (i.e., the record of what has been done to data) to support validation of results and post-hoc data analysis and visualization. Data and Visualization is the title for Data Management (DM) & Data Analytics and Visualization (DAV) activities in the Exascale project.

Data management (DM) activities address the severe I/O bottleneck and challenges of data movement by providing and improving storage system software; workflow support including provenance capture; and methods of data collection, reduction, organization and discovery.

Data analytics and visualization (DAV) are capabilities that enable scientific knowledge discovery. Data analytics refers to the process of transforming data into an information-rich form via mathematical or computational algorithms to promote better understanding. Visualization refers to the process of transforming scientific simulation and experimental data into images to facilitate visual understanding. Data analytics and visualization have broad scope as an integral part of scientific simulations and experiments; they are also a distinct separate service for scientific discovery, presentation and documentation purposes, as well as other uses like code debugging, performance analysis, and optimization.

The scope of activities falls into the following categories:

- Scalable storage software infrastructure – system software responsible for reliable storage and retrieval of data supporting checkpointing, data generation, and data analysis I/O workloads
- Data collection, reduction, and transformation – enabling complex transformation and analysis of scientific data where it resides in the system and as part of data movement, in order to reduce the cost to solution
- Data organization and discovery – indexing and reorganizing data so that relevant items can be identified in a time- and power-efficient manner, and complex scientific data analysis can be performed efficiently on Exascale datasets
- In situ algorithms and infrastructure – performing DAV while data is still resident in memory as the simulation runs enabling automatic identification, selection and data reduction for Exascale applications.
- Interactive post-hoc approaches – on data extracts that produced in situ and support post-hoc understanding through exploration.
- Distributed memory multi-core and many-core approaches, for the portable, performant DM and DAV at Exascale.

4.4.2 *Assumptions and Feasibility*

- Scaling up traditional DM and DAV approaches is not a viable approach due to severe constraints on available memory and I/O capacity, as well as dramatically different processor and system architectures being at odds with contemporary DAV architectures.

- Simulations will produce data that is larger and more complex, reflecting advances in the underlying physics and mathematical models. Science workflows will remain complex, and increasing requirements for repeatability of experiments, availability of data, and the need to find relevant data in Exascale datasets will merit advances in workflow and provenance capture and storage.
- The expense of data movement (in time, energy, and dollars) will require data reduction methods, shipping functions to data, and placing functionality where data will ultimately reside.
- Solid-state storage will become cheaper, denser, more reliable, and more ubiquitous (but not cheap enough to replace disk technology in the Exascale timeframe). Exascale compute environments will have in-system nonvolatile storage and off-system nonvolatile storage in addition to disk storage. Applications will need help to make use of the complex memory/storage architectures.
- Disks will continue to gain density but not significant bandwidth; disks will become more of a capacity solution and even less a bandwidth one.
- Industry will provide parts of the overall data management, data analysis and visualization solution, but not all of it; non-commercial parts will be produced and maintained.
- This plan and associated costs were formulated based on the past decade of DOE visualization and data analysis activities, including the successful joint industry/laboratory-based development of open-source visualization libraries and packages (VTK, VisIt, and ParaView).

4.4.3 Objectives

Data management, analysis and visualization software must provide the following:

- Production-grade exascale storage infrastructure(s), from application interfaces to low-level storage organization, meeting requirements for performance, resilience, and management of complex Exascale storage hierarchies
- Targeted research to develop a production-grade in situ workflow execution system, to be integrated with vendor resource management systems, meeting science team requirements for user-defined and system-provided provenance capture and retention
- Production-grade system-wide data transfer and reduction algorithms and infrastructure, with user interface and infrastructure for moving/reducing data within the system, to be integrated with vendor system services and meeting science and national security team requirements
- Production-grade metadata management enabling application and system metadata capture, indexing, identification, and retrieval of subsets of data based on complex search criteria and ensures that technologies target science and national security team requirements
- Targeted research to develop a production-grade in situ algorithms, to be integrated with open source visualization and analysis tools and infrastructure, meeting science team data reduction requirements
- Targeted research to develop a production-grade algorithms for the new types of data that will be generated and analyzed on Exascale platforms as a result of increased resolution, evolving scientific models and goals, and increased model and data complexity
- Targeted research to develop a production-grade post-hoc approach that support interactive exploration and understanding of data extracts produced by in situ algorithms
- Production-grade Exascale data analysis and visualization algorithms and infrastructure, meeting requirements for performance, portability and sustainability for evolving hardware architectures and software environments

4.4.4 *Plan*

Productization of technologies is a necessary step for adoption, research-quality software is not enough. One approach we will take is to fund vendors of products in related areas to integrate specific technologies into their product line. When developing objectives for this activity, a focus was placed on the availability of products that deliver these technologies on platforms of interest. Activities can be separated into two categories:

1. Community/Coordination: designed to build the R&D community, inform ourselves and the community regarding activities in the area, track progress, and facilitate coordination.
2. Targeted R&D: Filling gaps in critical technology areas (storage infrastructure, workflow, provenance, data reduction and transformation, and organization and discovery).

Portions of the DAV software stack are being turned into products and supported by industry, and that will help to control costs in the long term. Activities to achieve the DAV objectives are heavily dependent on developments across the Exascale project, and thus close coordination with other teams is essential. Close engagement with application scientists is crucial to the success of DAV, both in terms of understanding and addressing the requirements of science at scale and ensuring that computational scientists are able to adopt and benefit from the DAV deliverables.

Many objectives need initial research projects to define plausible solutions. These solutions will be evaluated and progressively winnowed to select the best approaches for the Exascale machine and the needs of science. Selected projects will continue to receive support to extend their research and development efforts to integrate their solutions into the open-source Exascale software stack.

4.4.5 *Risks and Mitigation Strategies*

There are specific risks identified for the Data and Visualization portfolio. These risks are tracked in the risk register.

- Application teams may continue to employ ad hoc methods for performing data management in their work, resulting in increased I/O bottlenecks and power costs for data movement. Application team engagement, working within the overall software stack, and input into HI will be necessary if results are to be deployed, adopted, and significantly improve productivity.
- Despite funding vendor activities, industry partners may determine the market is insufficient to warrant meeting Exascale requirements.
- If vendor integration and targeted R&D activities are not closely coordinated, gaps will not be effectively identified and targeted, or successful R&D will not be integrated into industry products in the necessary timeframe.
- Vendors supplying data management solutions are likely to be distinct from Exascale system vendors. Additional coordination will be necessary, beyond DM productization, in order to ensure interoperability of DM solutions with specific Exascale platforms.
- Data management from an application perspective is tracked in one of the identified risks. Additionally, the software stack tracks several risks indirectly related to data management as well.
- Failure of scientists to adopt the new DAV software is a major risk that is exacerbated if the DAV software is research quality. Mitigating this risk depends on close engagement with domain scientists and supporting layers of the software stack through co-design activities, as well as investment in development and productization of DAV codes.
- Redundant efforts in domain science communities and within ASCR-supported activities such as SciDAC result in wasted resources. Communication and close coordination provide the best strategy for mitigation.

- Fierce industry and government competition for DAV experts creates a drain on laboratory personnel in DAV and makes lab hiring in this area difficult. Stable funding and a workforce development program would help to mitigate these risks.
- A skilled workforce is required for a successful exascale project.

4.4.6 Future Trends

Graphics Architectures and Approaches Graphics architectures are improving in terms of raw computational power and through the addition of specialized libraries for accelerating ray-tracing, volume rendering, and denoising. Nvidia has added specialized hardware processing units for ray-tracing and machine learning to their GPU offerings. Intel has developed a suite of CPU accelerated libraries that support OpenGL (OpenSWR), ray-tracing (Embree, OSPRay), volume rendering (Open Volume Kernel Library) and de-noising (Open Image Denoise). From a visualization and rendering perspective, ray-tracing provides significantly improved rendered results over traditional scan-conversion based approaches. A near-term opportunity is to take advantages of such functionality for our rendering needs. Longer term, we will look into leveraging these hardware accelerated approaches to accelerate visualization and analysis tasks.

In-Situ Analysis and Automation A key thrust of the Data and Visualization area is the focus on in situ analysis in order to filter important information as it is being generated by the simulations. In addition to our algorithmic and infrastructure efforts, automatic techniques and workflows must be developed to guide the overall in situ analysis process.

Workflows Slowly, more complex workflows are becoming a more significant component of the job mix on ECP-relevant platforms, partially driven by the increased use of these systems for machine learning applications. Workflows can drive degenerate use cases in the storage stack, such as the use of the file system for communication between tasks, when tools from outside the HPC community are adopted without change. Alternative approaches to enable communication between tasks exist but must be adapted to facility contexts, and technical roadblocks (e.g., difficulty in communicating between separate jobs) must be overcome.

AI AI applications will appear more frequently in the job mix. This impacts the requirements for data storage, as new classes of data become more prominent in application input datasets. It also impacts technologies for understanding application behavior, as these jobs are often not using MPI, a common assumption in tool sets. Finally AI-focused applications do not exhibit the common pattern of alternating phases of I/O and computation seen in simulation codes, driving a need for attention on methods of I/O optimization that do not rely on explicit collective I/O phases.

Networks Network architectures are still in flux, and specific new technologies such as Slingshot from Cray will bring new capabilities such as more advanced congestion detection and mitigation that change how networks will behave in the face of mixed communication and I/O traffic or the impact of communication-heavy applications on other applications in the system, etc. Assumptions regarding how I/O traffic fits into this picture may need to be reexamined. The libfabric interface for accessing networks appears to be the most promising portable interface for use outside of MPI, and teams will need to assess how to best use libfabric across platforms of interest as well as possibly advocating for specific capabilities in libfabric that fall outside of traditional MPI use cases, such as the common pattern of clients connecting and detaching from long-running services.

Object stores Facilities are planning deployments of non-POSIX storage solutions. One of the first of these will be the DAOS deployment on the A21 system at Argonne. The DAOS interfaces are available for teams to begin to understand, an HDF5 front-end for DAOS is available, and there are some examples of DAOS use for scientific codes. It is likely that the highest performance will come from applications directly using the DAOS APIs, and work to allow understanding of how these APIs are used would be beneficial.

Compression Compression will continue to play an important role in computation as a vehicle for addressing the explosion in size of datasets and outputs. Improved integration of compression capabilities in libraries supporting parallel I/O will continue to be a topic for further development, and techniques for allowing concurrent updates while compression is enabled specifically need more exploration. The use of lower precision data types has the potential of speeding up the visualization and analysis process as well as reducing data sizes without significantly degrading the accuracy of results.

Storage technologies and architectures Even in systems that will continue to employ POSIX file systems as the main "scratch" store, the hardware on which these file systems are stored will be changing. For example, the Perlmutter system will provide a 30 PB nonvolatile storage tier using Lustre. The file system teams (e.g., Lustre team) will be working to maximize performance on these new storage back-ends, but simultaneously higher software layers must consider how this significant change impacts their assumptions about the relative costs of communication and data storage for common patterns of access.

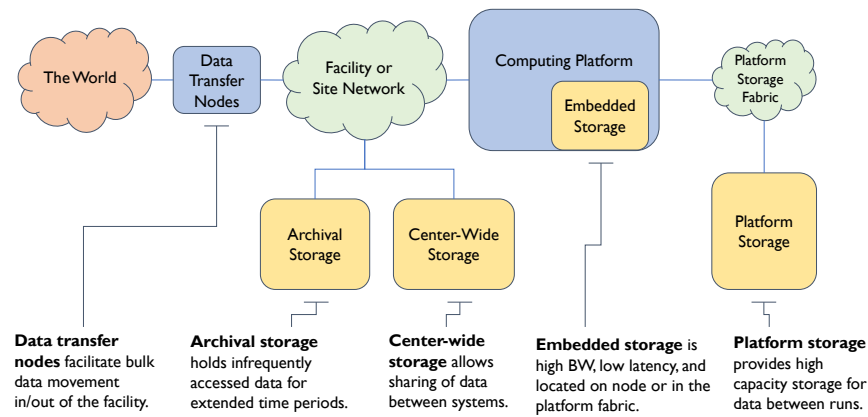


Figure 75: A notional diagram of DOE facility storage resources. Not all systems have each role filled, and often additional network connections exist to accelerate specific data flows.

Figure 75 depicts a notional diagram of the storage resources surrounding a leadership class platform at a DOE facility. In this diagram, “platform” is the HPC system itself: Theta, Summit, Cori, Frontier, Perlmutter, Aurora, etc.

Note that this is a notional diagram. There are often additional connections to speed specific transfers, and some sites augment one resource while omitting another. Data transfer nodes provide access to data stored on facility storage from the outside world: Typically this is enabled using GridFTP, Globus Online, htar, or similar.

Storage can play many roles:

- Archival storage holds *cold* data. Tape is still the dominant medium for archival storage, but disk is also used, both as cache and as permanent storage (sometimes spun down).
- Center-wide storage allows easy data access between systems. This might include home directories and some other shared data volumes. Often performance is limited compared to platform storage.
- Platform storage is connected to a limited number of platforms in the facility and is meant to be a high-performance, high-capacity store for data that will be used for multiple runs. While disk drives are still used in some platform storage deployments, increasingly solid-state storage (e.g., SSDs) is being employed in this role.
- Embedded storage is located very close to the platform itself, either as node-local storage or tightly integrated into the fabric. Technologies used include NVMe and SSDs. Embedded storage that is available across the system is perhaps best thought of as a special kind of platform storage.

Table 14: Storage system specifications for current platforms.

	ALCF Theta				NERSC Cori				OLCF Summit			
	Archive	Center	Platform	Embedded	Archive	Center	Platform	Embedded	Archive	Center	Platform	Embedded
SW	HPSS	Lustre	Lustre	ext3	HPSS	GPFS	Lustre	DataWarp	HPSS	GPFS	N/A	XFS
HW	LTO8 Tape/HDD	SSD/HDD	HDD	SSD	3592 Tape/HDD	HDD	HDD	SSD	3592 D Tape/HDD	HDD		SSD
Capacity	305 PB	200 PB	10 PB	549 TB	230 PB	128 PB	30 PB	1.8 PB	130 PB	250 PB		7.3 PB
BW	90 GB/s (cache)	650 GB/s	240 GB/s	6.6-9.2 TB/s	100 GB/s (cache)	100 GB/s	700 GB/s	1.7 TB/s	120 GB/sec (cache)	2.2-2.5 TB/s		9.7-27 TB/s
Usability	Medium	High (POSIX)	High (POSIX)	Low (per-node)	Medium (hsi, htar, ftp, globus)	High (POSIX)	High (POSIX)	Medium (POSIXy)	Medium (his, htar, globus)	High (POSIX)		Low (per-node)

Table 14 captures the salient characteristics of the storage deployments for the current generation of systems: Theta, Cori, and Summit. These systems largely reflect trends in DOE storage over the last decade: HPSS archival storage coupled with a POSIX center-wide file system provided by GPFS or Lustre and backed by hard disk drives (HDDs). In two cases, a faster, platform-specific POSIX file system is deployed, while at OLCF the team chose to instead concentrate on a very high performance center-wide file system that is available on other resources as well.

Of note in these systems are some embedded storage options that have provided the community with some early experiences with SSD storage. On Theta and Summit, local SSDs are available that have seen limited use by specific teams. On Cori, the DataWarp service allows for SSD-backed storage pools to be allocated that are visible to an entire job or workflow. This functionality is close to the model that users are accustomed to, and the resource has seen significant use.

Table 15: Projected storage specifications for upcoming platforms.

	ALCF Aurora				NERSC Perlmutter				OLCF Frontier			
	Archive	Center	Platform	Embedded	Archive	Center	Platform	Embedded	Archive	Center	Platform	Embedded
SW	HPSS	Lustre	DAOS	N/A	HPSS	GPFS	Lustre	N/A	TBD	Lustre	N/A	XFS
HW	LTO8 Tape/HDD	SSD/HDD	PM/NVME		3592 Tape/HDD	HDD	NVME			SSD/HDD		TBD
Capacity	305 PB	200 PB	220 PB		230 PB	200 PB	35 PB			700 PB		TBD
BW	90 GB/s (cache)	650 GB/s	25+ TB/s		100 GB/s (disk)	500 GB/s	5 TB/s			10 TB/s		TBD
Usability	Medium	High (POSIX)	? (non-POSIX)		Medium (hsi, ftp, globus)	High (POSIX)	High (POSIX)			High (POSIX)		Low (per-node POSIX)

Table 15 captures the current, public understanding of storage characteristics for the next generation of systems: Aurora, Perlmutter, and Frontier. A number of observations can be made from this data. First, POSIX will remain the dominant (at least low-level) API for interacting with the primary storage resources (embedded, platform, and center-wide). Obtaining peak performance will likely remain a challenge for users, but with node counts somewhat stalled, storage software scalability is not further pressured. Second, DAOS, will appear as a POSIX alternative. DAOS is an Intel product that will provide a form of globally accessible key-value style of storage. However, a POSIX “vener”, or compatibility layer, will be provided that will give users an easy way to make use of the resource. With all the major ST I/O libraries already having the ability

to implement alternative back-ends, the challenge will be more about how to persist things *outside* DAOS efficiently (e.g., converting back to POSIX files). Finally, there will still be a non-global, embedded storage resource on Frontier. The best ways to utilize this resource need to be studied, but systems like UnifyFS could play a role.

4.4.7 WBS 2.3.4.01 Data & Visualization Software Development Kits

DataViz SDK Overview The Data & Visualization (DataViz) SDK aims to create a production-quality infrastructure necessary to manage, share, and facilitate data analysis of mission-critical codes at scale. The project focuses on community development and a commitment to success via quality improvement policies, better build and deployment processes, and the ability to use diverse, independently developed DataViz SDK projects, in combination, for data analysis and visualization problems.

The DataViz SDK's responsibility is to coordinate the development, testing, and deployment activities for the suite of projects across the ECP Data & Visualization area as it applies to cross-project interoperability, and develop the necessary tooling and shared infrastructure to serve these goals. This coordination's resulting product is a unified set of usable, standardized, and interoperable packages ready for the upcoming exascale machines. We have designed the efforts to support the DataViz SDK to fit within the overarching goal of leveraging and integrating data management, analysis, and visualization techniques developed across the ECP ST ecosystem to support scientific discovery and understanding.

In addition, DataViz SDK provides a capability supporting collaborative analysis and visualization software development, helping independent teams accelerate the adoption of best practices, enabling interoperability of independently developed software, and improving developer productivity and sustainability of the software products.

Key Challenges Scientists and engineers from various research cultures and significantly different software engineering maturity levels develop Data & Visualization software packages. In addition to the challenges outlined in Section 4.5.7, ECP Data & Visualization software packages will use different combinations of dependency software in various configurations. Visualization applications and libraries, in particular, utilize lower-level graphics libraries from an OpenGL stack needing to be reliably mapped to a diverse set of underlying hardware. These applications demand the deployment infrastructure supporting the appropriate combination of software and hardware-based rendering on NVIDIA, AMD, or Intel GPUs and accelerated offscreen rendering APIs like EGL. Requirements such as these place the DataViz SDK between the hardware teams and the analysis and visualization software developers preparing to run on new architectures yet delivered by ECP.

DataViz SDK software packages also have, on average, a much deeper dependency tree than is typical within HPC. As of this writing, the optimized set of thirteen ECP DataViz SDK packages requires over 201 dependencies. Many packages share these dependencies, each with their own set of constraints. This combination presents a unique challenge to ensure compatibility, interoperability, and reliability of the entire stack as a whole, beyond the individual packages.

Solution Strategy First, the DataViz SDK solution strategy involves pursuing usability, standardization, interoperability, and sustainability goals through a set of community policies to improve software practices. The DataViz SDK community policy tasks have required us to define a common terminology for effective communication.

Second, we leverage shared infrastructure, such as the Spack [1] package manager and CI testing at ECP facilities. We have built the SDK release and delivery deployment goals on Spack as a unifying package manager, while our reliability and sustainability goals benefit from and leverage a combination of CI testing infrastructure from upstream Spack, DOE facilities, and dedicated project resources.

Finally, we define a spack meta-package, `ecp-data-vis-sdk`, to enable the delivery of ECP targeted configurations of data and visualization packages through E4S. The meta-package establish dependencies for the packages within the DataViz SDK, and serves as the backbone of our interoperability testing and deployment efforts. It enables the coincident optimized deployment of the diverse set of packages in the ECP Data & Visualization portfolio, including visualization tools and libraries (e.g., ASCENT, Catalyst, Cinema,

ParaView, VTK-m) data-reduction and compression libraries, (e.g., SZ, ZFP), and I/O and data service libraries (e.g., ADIOS, Darshan, HDF5, PNetCDF, UnifyFS, VeloC).

The packages contained in the meta-package are able to be deployed in a way that enables all of the appropriate features for each package when the others are present and in a way that satisfies one another's dependencies. The release strategy in the early efforts of the SDK was to push product readiness for inclusion in the E4S releases by assisting individual packages with Spack packaging and CI testing. With that effectively achieved in the early efforts of the SDK our focus has now shifted to the collective interoperable deployment and testing of all the packages on targeted ECP platforms of interest.

Recent Progress Recent progress, including integration with the upstream Spack CI and deployment on Spock, is described below.

Integration with upstream Spack CI The spack package manager is the distribution mechanism of choice for ECP. To ensure reliability for its packages the spack project has developed a CI capability for specific package collections of interest to be built and validated in every pull request. Through coordination with the spack project, the DataVis SDK has integrated its spack meta-package as one of the primary pipelines of interest for the spack project. As a result, the packages and dependencies enabled by the DataVis SDK are validated for build correctness on every change and addition to spack.

Deployment on Spock The Frontier system at OLCF is scheduled to come online by the end of 2021 or early 2022. In preparation we have targeted Spock, the testing and development environment for Frontier, for an initial full deployment of the DataVis SDK. We have now successfully been able to deploy the `ecp-data-vis-sdk` meta-package on the Spock system using the Cray Programming Environment with both GNU and Cray compilers while enabling twelve distinct Data & Vis packages. This will minimize the effort needed to deploy on Frontier when it comes online.

Next Steps We highlight our next steps in the follow on project milestones.

STDV01-38 Implement and test accelerator variants in the SDK: The initial SDK package lacks the necessary features to enable and propagate GPU features for the packages. We will implement and test GPU options in the SDK for each of the accelerator platforms of interest (AMD, NVIDIA, and Intel) and ensure they are consistently and reliably applied to the DataVis SDK packages.

STDV01-39 Deployment of the SDK into E4S: We will integrate the `ecpdata-vis-sdk` spack meta-package into the E4S distribution and it will become the provider-of-choice for the ECP Data & Visualization packages it supports. This will ensure that E4s will provide the most effective and useful configurations of these packages.

STDV01-40 Deployment of the SDK in facility environments: We will deploy the full SDK on the various ECP platforms available in production. This will ensure that when Data & Vis packages are provided via spack by facilities then they are in the most effective and useful configuration possible for their users.

HDF5

Overview HDF5 is a data model, I/O library, and file format to store and manage data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O for high volume and complex data. Its extensive ecosystem is illustrated by over 2,000 repositories on GitHub depending on HDF5 or 160 packages in Spack along with many third-party tools that support HDF5, e.g., MATLAB, and open source packages, e.g., h5py, Pandas, Keras, and TensorFlow.

HDF5 is portable and is extensible, allowing applications to evolve in their use of HDF5. It supports many different types of data stores, such as local, clustered, and networked file systems and object stores. HDF5 is highly customizable through its purposefully designed extension interfaces, Virtual Object Layer (VOL) and Virtual File Driver (VFD) interfaces as shown in Figure 76.

The most recent extensions include support for Intel DAOS high-performance storage and NVidia's GPUDirect interface for data movement between storage and GPU memory. Numerous ECP applications use

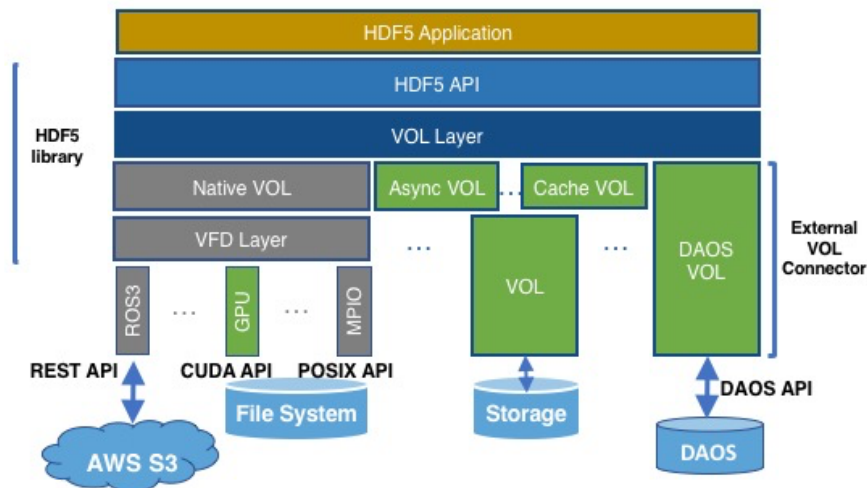


Figure 76: HDF5 architecture: green rectangles represent dynamically loaded VFD and VOL connectors to access data on different storage devices; grey and blue rectangles represent components of the HDF5 library.

HDF5 or high-level libraries built on top of HDF5 (for example, H5Part) for data management. Therefore, HDF5 is a critical part of the Data & Vis SDK.

The HDF Group team’s responsibility is to maintain HDF5 on the ECP systems, develop enhancements to boost the performance of the ECP HDF5 applications; integrate enhancements developed by other ECP teams (e.g. DataLib, ExaIO); and deliver HDF5 to the ECP users as a part of the Spack package and Data & Vis SDK.

Key Challenges The HDF Group team faces two major challenges. The first challenge is extending HDF5 legacy software to deliver high-performing features for the Exascale systems while maintaining HDF5 current performance standards and avoiding HDF5 software disruption for the millions of non-HPC users. The second challenge is the complexity of the HDF5 software that may lead to its misapplication, resulting in poor performance, especially in the HPC environment. The third challenge is delivering contributions to HDF5 created by other teams promptly and according to HDF5 regression testing and coding standards.

Solution Strategy To lower the barrier for adopting HDF5 by ECP applications and delivering new features timely, The HDF Group (1) provides HDF5 software that is regularly tested on and released for ECP platforms, (2) integrates newly developed features into mainstream HDF5 and makes them promptly available via Spack and Data & Vis SDK, (3) educates HDF5 users on the best HDF5 practices and new features, and (4) works closely with ECP teams on features development and applications’ tuning.

Recent Progress In 2021, The HDF Group developers made enhancements to the HDF5 library to address requirements of the HDF5 ExaIO VOL connectors - [Async](#), [Pass-through](#) and [Cache](#) VOL connectors, and [NVIDIA GPU Direct I/O VFD](#). The requested changes are available in the [HDF5 develop branch](#) on GitHub and will be released in the HDF5 1.13 series of releases and later. The releases can be obtained from The HDF Group [website](#) or using the Spack package manager.

We performed a major refactoring of internal HDF5 library APIs to accommodate new [asynchronous APIs](#). New HDF5 APIs are designed for HDF5 VOL connectors that support asynchronous operations using the [HDF5 Event Set APIs](#). This feature allows I/O to proceed in the background while the application performs other tasks.

HDF5 VOL and VFD APIs were reworked to address the issues brought up by the ExaIO developers. We held a webinar on the changes to help HDF5 VOLs and VFDs developers with porting existing connectors to

the new version of the HDF5 library. Recording and slides are available from [The HDF Group website](#). ECP ExaIO VOL connectors were integrated with Spack. The HDF Group also worked with the ExaIO team on coordination of the HDF5 library and VOL connectors releases.

CI testing was set up to keep HDF5 VOL connectors development and the HDF5 library development in sync. During our integration and testing work, we addressed several issues and desired improvements reported by the DataLib and ExaIO teams. We added a feature that allows auto-detection of a VOL connector used to open an HDF5 file; the feature is available in HDF5 1.13.0 and HDF5 1.12.2 and later maintenance releases; we made improvements to collective metadata write operations and addressed the issues found when using IBM SpectrumScale MPI on the Summit system.

The required changes to the HDF5 library architecture for supporting HDF5 VOL connectors added 7-10% overhead for the applications that do not use connectors at all. The overhead was especially prominent for applications that work with a lot of HDF5 objects. We identified and implemented several optimizations and saw up to 10% reduction in time spent by a [benchmark](#) provided to us by LLNL.

Other integration work includes support for SZ and ZFP compression filters in the maintenance releases of the HDF5 library starting with the HDF5 1.10.7 release. The HDF Group developers also investigated HDF5 performance with SZ and ZFP HDF5 compression filters. Our study showed that HDF5 filters pipeline implementation doesn't create additional overhead for SZ and ZFP compressions.

In 2021, The HDF Group continued with [outreach activities](#). Recent activities included two Tutorials "[HDF5 Application Tuning](#)" and "[New features in HDF5 1.13.0](#)," and a [white paper](#). The HDF team delivered HDF5 tutorial for ECP 2021 annual meeting and gave a talk on the new ECP VOL/VFD features at the HDF5 BoF at the annual meeting.

Preliminary Experiences on Early Access systems The HDF5 software has been tested regularly with and without Spack on Arcticus, Theta, ThetaGPU, Spock, Crusher and Perlmutter. In the absence of GitLab testing framework, we have set up our own custom automated regression testing on Theta and Crusher with results posted to [CDash](#). We will extend this approach to other EAS until GitLab is available. We didn't encounter any particular problems with porting and testing our software. Along with HDF5 software, we also ported and tested [HDF5 VOL connectors](#) created by ECP teams on targeted EAS. The [Cuda GPU VFD](#) was tested on ThetaGPU and is available in the `hdf5-vfd-gds` Spack package. It was discovered in testing that running tests with `hdf5-vfd-gds` on ThetaGPU required setting the Spack stage directory to the Lustre filesystem in order for the test to pass. Applications using CUDA GPU VFD should be run only on a parallel filesystem.

Next Steps The HDF Group activities will continue in two areas: productization of the HDF5 features developed for ECP applications and outreach. In 2022, we will continue with CI testing on ECP systems and will provide maintenance releases of the HDF5 library. We will deliver a TOOLKIT for the VOL connectors developers to facilitate the development of new HDF5 connectors. We will integrate HDF5 subfilng VFD and multi-dataset feature (access data in multiple datasets using a single I/O call) that we are developing under the ExaIO project and release it in the mainstream HDF5. We will continue working with the ECP HDF5 applications teams on I/O performance, conduct Tutorials and Webinars, and create additional documentation on efficient usage of HDF5 in the ECP HPC environment.

4.4.8 WBS 2.3.4.09 ADIOS

Overview The Adaptable I/O Systems (ADIOS) [194, 195] is designed to tackle I/O and data management challenges posed by large-scale computational science applications running on DOE computational resources. ADIOS has dramatically improved the I/O performance of petascale applications from a wide range of science disciplines, thereby enabling them to accomplish their missions. The ADIOS ECP project is working on goal of transforming the ADIOS 1.x version, which has been used successfully on Petascale resources into a tool that will efficiently utilize the underlying Exascale hardware, and create a community I/O framework that can allow different ECP software to be easily plugged into the framework. The cornerstone of this project are to 1) efficiently address Exascale technology changes in compute, memory, and interconnect for Exascale applications; 2) develop a production-quality data staging method to support Exascale applications and software technologies that require flexible in situ data reduction and management capabilities; and 3)

use state of the art software engineering methodologies to make it easier for the DOE community to use, maintain, and extend ADIOS. More precisely, our aim is to develop and deploy a sustainable and extensible software ecosystem. To make this into an ecosystem (rather than a point solution), this effort must result in an infrastructure that can be used effectively, customized, and shared among a variety of users, Exascale applications, and hardware technologies. Technically, we are achieving this goal by: refactoring ADIOS with the goal of improving modularity, flexibility, and extensibility by using C++; and extending, tuning, and hardening core services, such as I/O and staging that supports Exascale applications, architectures, and technologies.

Key Challenges The core challenge of ADIOS is in its name—adaptability. In order to present a uniform user interface while also being able to harness the performance improvements available in the wide variety of storage and interconnect capabilities, the internal structure of the ADIOS framework must address a number of portability, functionality, and performance tuning challenges. The internals should be constructed so that with no more than a small flag or runtime configuration a science code can move from doing I/O into a large Lustre parallel file system (with automatic calculation of file striping and number of files per directory) to utilizing burst buffer storage (with controls for delayed synchronization between the buffer and an archival store) or feeding the data directly into a concurrent application

The challenge of supporting hardware portability and runtime performance tuning also impose a third related challenge for software engineering of the system. In order for the code to be sustainable in the long term, while also offering guarantees of service to the end user, requires special attention to the architecture of the code base. The consequences of trying to address these three challenges, hardware portability, runtime performance, and sustainable engineering, have driven our approach and deliverable schedule for ADIOS in ECP.

Solution Strategy The ADIOS effort has two primary thrusts:

1. Scalable I/O: ADIOS has a data format designed for large scale parallel I/O and has data transport solutions to write/read data from/to the storage system(s) efficiently.
2. Scalable data staging support: ADIOS includes data transport solutions to work on data in transit, that is, to move data memory-to-memory, from one application stage to another without using file system I/O.

The challenges of portability and performance apply for both of these thrusts; to a certain extent, the third challenge around software engineering emerges from the need to support these two very different categories under a single user interface. Capitalizing on the experiences and performance expertise from our initial ADIOS platform, the ECP project wraps and extends this functionality to make it more sustainable, maintainable, and hopefully also more approachable for a wide community of users and developers. The project approach focuses on doing deep dives with end scientist users and developers in order to make sure that the computer science development process leads to specific, verifiable results that impact the customers.

Recent Progress A new version of the Application Programming Interface unifies staging I/O and file I/O [196], and the new, object-oriented, code framework [197] supports writing and reading files in two different file formats (ADIOS BP format and HDF5 format) and in situ with different staging implementations for various use cases. The new framework focuses on sustainable development and code reusability. The team also created the new scalable staging transport learning from the many lessons from using ADIOS for data staging and code coupling by applications in the past. As can be seen in Figure 77, this past experience with methods and deep science engagements has led to demonstrations at leadership computing scale (on Titan and Summit).

The new design focuses on stability and scalability so that applications can rely on it in daily production runs just as they have relied on the high performance file I/O of ADIOS. The new code base is governed with state-of-the art software development practices, including GitHub workflow of Pull-Requests with reviews, continuous integration that enforces well-tested changes to the code only, and nightly builds to catch errors on various combinations of architecture and software stack as soon as possible. Static and dynamic analysis are integrated to the GitHub workflow to catch errors before they cause trouble. Code coverage tools also help

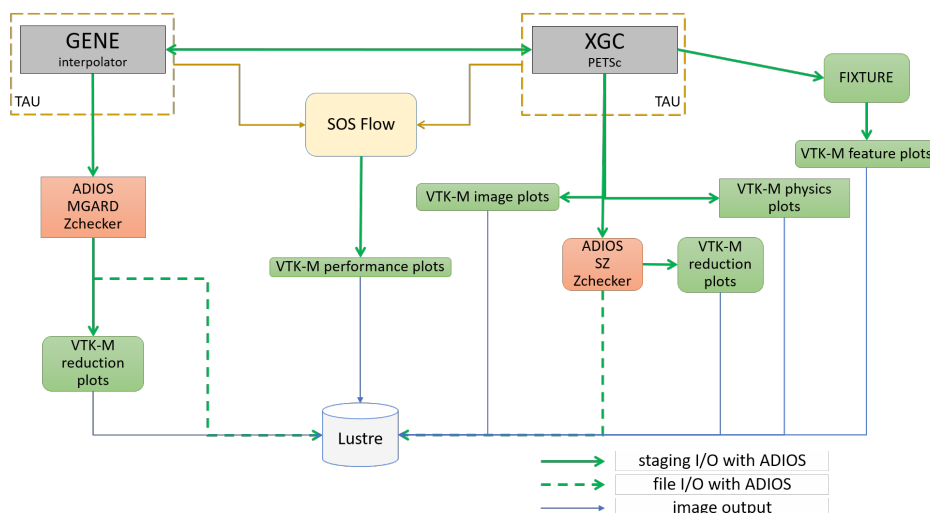


Figure 77: An example of using ADIOS to support ECP science. This sketch represents the demonstration at the February 2018 ECP Meeting, which featured WDM Fusion, CODAR, ADIOS, and other joint ECP activities. Note that all of the green arrows in the figure represent data communication or storage handled by the ADIOS infrastructure.

with increasing code quality. The team has access to and the code is continuously tested on DOE machines (Summit, Cori and Theta) using several ECP application codes and realistic science simulation setups (e.g. for WDMApp, E3SM-MMF and EXAALT application setups).

For interoperability with the other main I/O library used in the ECP program, HDF5, we have added compatibility in various ways. ADIOS has an engine to write/read HDF5 files using the original HDF5 library linked with ADIOS. A user can just change an option to switch from ADIOS-BP output to HDF5 output. On the other hand, ADIOS provides an HDF5-VOL layer, so that an HDF5 application can choose ADIOS as the underlying driver to write ADIOS-BP files from an application using HDF5.

Early Access Systems We have built ADIOS on EAS systems of Frontier and Aurora. For Frontier, we need to update ADIOS to work properly with the CXI libfabric provider from Cray, which is a new provider not seen before on existing systems. We are also developing the support for GPU-aware I/O on the AMD GPUs. For Aurora, we have developed a DAOS transport for ADIOS I/O to be able to write/read data on the DAOS filesystem.

Next Steps In the sixth year of the project we will be focusing on testing, scaling and optimizing ADIOS (storage IO, in situ and code coupling services) on the new Frontier exascale computer, with its multi-tier Lustre storage system, local-node storage, and the Slingshot network. We will primarily focus to deliver ADIOS services to our ECP application partners working on their KPP-1 goals on Frontier.

4.4.9 WBS 2.3.4.10 DataLib

Overview The Data Libraries and Services Enabling Exascale Science (DataLib) project has been pushing on three distinct and critical aspects of successful storage and I/O technologies for ECP applications: enhancing and enabling traditional I/O libraries used by DOE/ECP codes on leadership platforms, establishing a nascent paradigm of data services specialized for ECP codes, and working closely with facilities to ensure the successful deployment of our tools. In FY20-23 we plan to continue to focus on these three complementary aspects of storage and I/O technologies, adjusting in response to changing needs and bringing these three aspects together to provide the most capable products for end users. DataLib activities ensure that facilities have key production tools, including tools to debug I/O problems in ECP codes; enable multiple I/O middleware

packages through Mochi and ROMIO; and will provide high performance implementations of major I/O APIs in use by ECP codes.

We strongly support ECP management’s shift of focus towards Hierarchical Data Format (HDF). In response to ECP guidance to prioritize the HDF5 API, we propose to emphasize enhanced HDF5 capabilities for ECP codes on current and future DOE leadership platforms, strengthening HDF as a core technology for the future. We propose to shift our focus away from ROMIO and PnetCDF development work to enable rapid progress on this topic. We will continue to support the use of Mochi tools for development of data services and I/O middleware, including assisting other ECP AD, ECP ST, and vendor teams in providing the best storage services possible for ECP applications. We will also continue to work closely with the facilities to ensure the availability and quality of our tools on critical platforms.

The Darshan I/O characterization toolset is an instrumentation tool deployed at facilities to capture information on I/O behavior of applications running at scale on production systems. It has become popular at many DOE facilities and is usually on by default. Darshan data dramatically accelerates root cause analysis of performance problems for applications and can also (in some cases) assist in correctness debugging. Our work in this project focuses on extending Darshan to new interfaces and ensuring readiness on upcoming platforms.

The ROMIO and Parallel netCDF (PnetCDF) activities focus on existing standards-based interfaces in broad use, assisting in performance debugging on new platforms and augmenting existing implementations to support new storage models (e.g., burst buffers). In addition to being used directly by applications, ROMIO and PnetCDF are also indirectly used in HDF5 and netCDF-4. Our work is ensuring that these libraries are ready for upcoming platforms and effective for their users (and ready as alternatives if other libraries fall short).

The Mochi and Mercury software tools are building blocks for user-level distributed HPC services. They address issues of performance, programmability, and portability in this key facet of data service development. Mercury is being used by Intel in the development of their DAOS storage service and in other data service activities, while within ECP the HXHIM and UnifyCR projects also have plans to leverage these tools. In addition to working with these stakeholders and ensuring performance and correctness on upcoming platforms, we are also working with ECP application teams to customize data services for their needs (e.g., memoization, ML model management during learning). These are supporting tools that are not represented as official products in the ECP ST portfolio.

Key Challenges Each of these subprojects has its own set of challenges. Libraries such as HDF, ROMIO, and PnetCDF have numerous users from over a decade of production use, yet significant changes are needed to address the scale, heterogeneity, and latency requirements of upcoming applications. New algorithms and methods of storing data are required. For Darshan, the challenge is to operate in a transparent manner in the face of continuing change in build environments, to grow in functionality to cover new interfaces while remaining lean from a resource utilization perspective, and to interoperate with other tools that use similar methods to hook into applications. Mochi and Mercury are new tools, so the challenge in the context of these tools is to find users, adapt and improve to better support those users, and gain a foothold in the science community.

Solution Strategy We have a solution strategy for HDF enhancement, supporting ECP applications and facilities, supporting data services, and integration and software QA.

HDF Enhancement HDF is the most popular high-level API for interacting with storage in the DOE complex, but users express concerns with the current The HDF Group (THG) implementation. We propose to perform an independent assessment and systematic software development activity targeting the highest possible performance for users of the HDF5 API on ECP platforms of interest.

Directly Supporting ECP Applications and Facilities We currently have ongoing interactions with E3SM (PnetCDF), CANDLE (FlameStore/Mochi), and ATDM/Ristra (Quantaii/Mochi), and we routinely work with the facilities as relates to Darshan deployments. Our work with these teams is targeted on specific use cases that are inhibiting their use of current pre-exascale systems, such as E3SM output at scale using

the netCDF-4/PIO/PnetCDF preferred code path. We will continue to work with these teams to address concerns, to maintain portability and performance, and may develop new capabilities if needs arise.

Supporting Data Services Mochi framework components are in use in multiple ECP related activities, including in the UnifyCR and Proactive Data Containers (PDC) in ExaHDF5 (WBS 2.4.x) and in the Distributed Asynchronous Object Storage and other services in the Intel storage software stack. The VeloC and DataSpaces teams (WBS 2.4.x and x.y.z as part of CODAR, respectively) are also strongly considering adoption of our tools. Mochi components enhance the performance, portability, and robustness of these packages, and our common reliance on Mochi components means that as Mochi improves, so do all these users.

Integration and Software QA DataLib has actively pursued integration with the ECP ST software stack through the development and upstreaming of Spack packages and the development and deployment of automated testing for DataLib technologies, so we are already well positioned in this aspect of our work. We anticipate this effort to continue throughout the FY20-23 timeframe, with the addition of pull requests submitted to THG to upstream HDF5 enhancements and effort applied to address identified issues in our technologies as appropriate.

Recent Progress We have made progress toward the HDF5 VOL plug-in and I/O improvements in E3SM, xRAGE, and CODAR along with reducing Darshan overhead and improving UCX support for Mercury.

HDF5 VOL Plug-in Deliver design of HDF5 VOL plug-in and improve I/O capabilities for ECP application. HDF5 VOL design is complete, and we have an initial prototype with which we are performing early performance testing. The design document and developer notes will be attached to the end of this report for convenience.

As a reminder, our HDF VOL implementation layers on top of the native VOL implementation provided by HDF. We capture write operations as a log of changes, and currently we persist both the description of the writes (in our documentation simply the metadata, stored in the metadata table) and the contents of the writes (in our documentation the log data, stored in the log dataset) as HDF datasets. Additionally, another table (the offset table) stores the offsets of specific datasets in the metadata table, allowing one to skip over unrelated datasets when looking for specific log entries. We're investigating the storage of metadata in memory, for performance reasons.

New optimizations have been made in the HDF VOL implementation, notably compression of dataset metadata, that allow this implementation to perform faster than our own PnetCDF implementation (and much faster than default HDF5 or netCDF-4).

I/O Improvement in E3SM We have previously extracted an I/O kernel from the E3SM code, and we are now working on a branch of this code that can write directly into HDF5. As background, the netCDF4 API is missing a key capability to describe I/O to multiple datasets as a single operation, and this limits the performance of netCDF4 for many scientific codes regardless of the underlying I/O API being used (e.g., PnetCDF, HDF). The E3SM team is aware of this deficiency, but to our knowledge there is no plan to address it at this time. Meanwhile, the HDF Group has been looking at multi-dataset writes of this type, although it is unclear when the capability might be made part of a production release. Never the less, working around the netCDF4 deficiencies allows us to isolate HDF performance/API challenges from the higher-level netCDF4 ones, and allows us to better explore our HDF VOL implementation.

Recently we have evaluated blob style storage for E3SM, which is a method of storing data without transforming into the global ordering desired for analysis. We have found that this method is competitive with ADIOS results seen elsewhere, reducing the need for the E3SM team to continue development of new ADIOS options.

I/O Improvement in xRAGE Many applications use HDF5 to write extremely large single files used for checkpoint restart and/or data analysis. Even on pre-exascale systems such as Sierra it can be difficult to achieve high performance (bandwidth) for these workloads. Applications writing single shared files must carefully align I/O operations to avoid file locking overheads and balance I/O operations across multiple writers to ensure high performance. To help diagnose issues and optimize I/O in production Sierra applications

the DataLib team has been working closely with LANL scientists to employ Darshan analysis capabilities that capture more detailed information about how HDF5, MPI-IO and POSIX are being used by the application and lower level software stacks. LANL has begun using Darshan on Sierra for a variety of applications including the xRAGE application and have worked with the DataLib team to identify potential bottlenecks in this and other applications.

I/O Improvement in CODAR Early in the year, we initiated discussions with the CODAR team building Chimbuko, an effort to create a provenance and performance analysis service for HPC. Through discussion on their use case, we determined that a new Mochi microservice was needed. The Sonata microservice has been developed to provide convenient storage and processing of their JSON record data. We continue to work with the CODAR team to address scalability concerns and enable their Chimbuko service to hit production goals.

Reducing Darshan Overheads Enhance HDF VOL implementation and evaluate Darshan and Mochi performance. We will revisit Darshan overheads on flagship platforms to ensure correctness and low-overhead operation, and we will incorporate new enhancements into the HDF VOL implementation for further evaluation. We will also perform a performance evaluation of key Mochi use cases on available test hardware.

This work uncovered some overheads in how we capture timing information on platforms such as Theta. Workarounds were developed to keep Darshan overheads in the noise for all but the most degenerate I/O use cases (e.g., single-byte writes via stdio).

UCX Support for Mercury We delivered UCX plug-in for Mercury in support of ECP applications and services: a tested UCX plug-in for Mercury, tuned for performance on ECP relevant platforms (e.g., Summit test systems), with appropriate nightly tests. Our colleagues working on DAOS have begun testing with this implementation with promising early results.

Next Steps Our near-term plan includes the following actions.

STDM12-36 The ROMIO team will implement a new plug-in for UnifyFS, revisit the ROMIO implementation overall to reduce synchronization costs, and work with the HDF5 and UnifyFS teams to benchmark and document the performance of these optimizations and possible next steps.

STDM12-37 We will evaluate and improve HDF VOL performance on latest available platforms, re-architect Darshan analysis tools using Python to be both more user friendly and more amenable to learning methods, and enhance Mochi benchmarks and improve performance for latest platforms based on STDM12-34 analysis.

STDM12-38 We will prepare materials and engage in outreach for DataLib software at ECP Annual or similar event, refine and enhance Darshan DAOS and HDF modules on latest available platforms, deep dive on performance of ATDM code (TBD) with HDF VOL, and develop new PnetCDF benchmark in conjunction with ECP user and analyze performance.

4.4.10 WBS 2.3.4.13 ECP/VTK-m

Overview As exascale simulations generate data, scientists must extract information and understand their results. One of the primary mechanisms for understanding these results is producing visualizations that can be viewed and manipulated. The VTK-m project is developing and deploying a scientific visualization library for exascale machines. This library can be used directly by simulation codes or by other visualization products, and VTK-m is embedded in other Exascale Computing Project (ECP) solutions and is the sole project providing support for exascale architectures. VTK-m supports features such as the shared-memory parallelism available on many-core CPUs and GPUs by redeveloping, implementing, and supporting necessary visualization algorithms.

One of the biggest recent changes in high-performance computing is the increasing use of accelerators. Accelerators contain processing cores that independently are inferior to a core in a typical CPU, but these cores are replicated and grouped so that their aggregate execution provides a very high computation rate at a much lower power. Current and future CPU processors also require much more explicit parallelism. Each successive version of the hardware packs more cores into each processor, and technologies such as

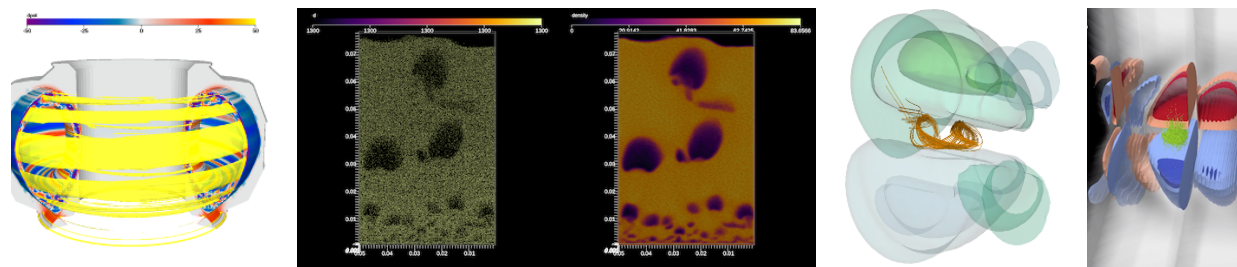


Figure 78: Examples of recent progress in VTK-m include (from left to right) in situ visualization with WDMApp simulations, particle density estimation, flow in laser wakefields, and in situ visualization with WarpX simulations.

hypertreading and vector operations require even more parallel processing to leverage each core’s full potential.

The VTK-m team is providing general-purpose scientific visualization software for exascale architectures that supports shared memory parallelism and fine-grained concurrency. The team is focused on providing abstract models for data and execution that can be applied to a variety of algorithms across many different processor architectures along with necessary visualization algorithm implementations. The results of this project will be delivered in tools currently used around the world today, such as ParaView and VisIt, as well as in new tools, such as Ascent, and in a stand-alone form.

Key Challenges The scientific visualization research community has been building scalable HPC algorithms for over 20 years, and today there are multiple production tools that provide excellent scalability. However, our current visualization tools are based on a message-passing programming model. More to the point, they rely on a coarse decomposition with ghost regions to isolate parallel execution [198, 199]. However, this decomposition works best when each processing element has on the order of a hundred thousand to a million data cells [200] and is known to break down as we approach the level of concurrency needed on modern accelerators [201, 202].

The VTK-m project addresses this challenge by providing the core capabilities to perform scientific visualization on exascale architectures, which can then be used within current tools, used by new libraries for in situ processing, or used directly by simulation codes. This fills the critical feature gap of performing efficient visualization and analysis on many-core CPU and GPU architectures.

Solution Strategy The ECP/VTK-m project leverages VTK-m [203] to overcome these key challenges. VTK-m has a software framework that provides the following critical features.

Visualization Building Blocks VTK-m contains the common data structures and operations required for scientific visualization. This base framework simplifies the development of visualization algorithms [204].

Device Portability VTK-m uses the notion of an abstract device adapter, which allows algorithms written once in VTK-m to run well on many computing architectures. The device adapter is constructed from a small but versatile set of data parallel primitives, which can be optimized for each platform [205]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [206, 207, 208] and while still providing performance comparable to less portable solutions [209]. Within the device adapter we are leveraging Kokkos [210] to rapidly port to ECP hardware.

Flexible Integration VTK-m is designed to integrate well with other software. This is achieved with flexible data models to capture the structure of applications’ data [211] and array wrappers that can adapt to target memory layouts [212].

Recent Progress The VTK-m project is organized into many implementation activities. The following features have been completed in the FY21 fiscal year.

VTK-m Releases VTK-m 1.6 was released in May 2021.

Kokkos Device adapters in VTK-m can now leverage the Kokkos programming model to more rapidly port to ECP hardware.

WDMApp Using specialized data adapters, VTK-m is able to link into the code coupling framework used by WDMApp to provide real-time in situ visualization as demonstrated in Figure 78.

Lagrangian Basis Flows By integrating VTK-m’s particle advection code, the VTK-m and ALPINE teams demonstrated high performance with significant data reduction by computing lagrangian basis flows for subsequent flow visualization [213, 214].

Particle Density Estimation Finding structures like bubbles in mesh-free data requires estimating the density of free-floating particles. VTK-m can now quickly estimate the density distribution of particles over a volume as demonstrated in Figure 78.

Laser Wakefield Flow Particles in a laser wakefield move with physics particular to the electromagnetic fields. VTK-m’s flow visualization is customized to properly compute particle paths in laser wakefield simulations from WarpX as shown in Figure 78.

WarpX As the underlying implementation of geometry processing and rendering in Ascent, VTK-m provides in situ visualization to WarpX simulations as shown in Figure 78.

Preliminary Experiences on Early Access systems The ECP/VTK-m team has made significant progress toward supporting visualization on the Spock early access system for Frontier during FY21. At the beginning of FY21, the team could compile a VTK-m smoke test with HIP and run it on an AMD GPU. This demonstrated running a VTK-m worklet (the encapsulation of a GPU kernel) on the GPU. This required resolving lots of special templating within the VTK-m software and was an achievement in its own right.

However, at the beginning of FY21, most of the features of VTK-m had to be disabled. This included mesh structures, filtering, and rendering, which comprises the major functionality implemented within the toolkit. FY21 included a major overhaul of the problematic features that were preventing operation on AMD hardware. The biggest change was the removal of virtual methods. VTK-m’s use of virtual methods was problematic for all GPU types and some required features were not supported on AMD. The removal of virtual methods required careful consideration of compiled data types that could no longer be hidden behind virtual method interfaces. New techniques to reduce template code paths needed to be designed. Another big overhaul was in the classes managing arrays. Array handles were redesigned to use buffer management of un-typed arrays. This moved much of the array management into libraries where they did not have to be recompiled for every use of every type. It also provided some more efficient ways to manage the allocation and transfer of data.

In addition to these internal changes, the VTK-m team worked closely with the Kokkos team and had regular meetings with AMD engineers. Progress required changes to VTK-m, Kokkos, HIP, and compilers. Below is a selection of links to software updates that were required.

- <https://github.com/kokkos/kokkos/issues/3431>
- https://gitlab.kitware.com/vtk/vtk-m/-/merge_requests/2282
- https://gitlab.kitware.com/vtk/vtk-m/-/merge_requests/2276
- <https://github.com/ROCm-Developer-Tools/HIP/pull/2183>
- <https://github.com/spack/spack/pull/19816>
- <https://github.com/kokkos/kokkos/issues/3581>
- <https://github.com/kokkos/kokkos/pull/3953>
- <https://gitlab.kitware.com/cjy7117/vtk-m/-/tree/hip-support>
- <https://reviews.llvm.org/D112733>

Currently, VTK-m with its test suite can be compiled on Spock. There are some caveats, however. It requires a ROCm 5.0 pre-release and the latest head of Kokkos. There is also currently an issue with linking the rendering library, which is being looked into.

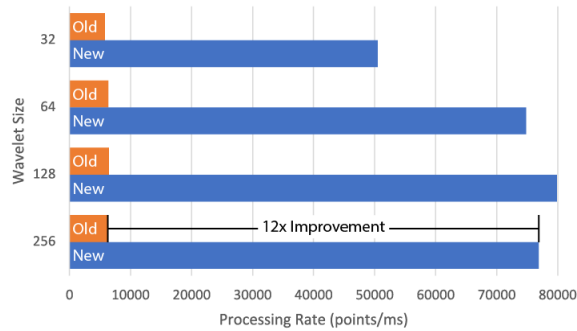


Figure 79: Performance improvement of VTK-m’s unstructured gradient benchmark under Kokkos.

The VTK-m team has also been working with the Kokkos team to improve performance while using Kokkos to drive devices, which is necessary for AMD GPUs. During this time, a performance issue for estimating gradients in unstructured grids was found. The VTK-m and Kokkos teams worked together to identify a register spilling problem in one of the GPU kernels. The kernel launch parameters for this case, and a 12-fold improvement for this algorithm was observed on Spock.

Next Steps The VTK-m team is focused on ensuring the success of our KPP-3 metric and will take the following next steps.

Satisfy Needs of ECP Applications The VTK-m team’s KPP-3 milestones hinge on applications using VTK-m code in a useful way. To ensure that VTK-m is satisfying the needs of ECP applications, the team will making modifications as needed. The initial focus will be with the WarpX, WDMApp, MFiX-Exa, and Nyx applications.

Porting to Exascale Architecture As Frontier becomes available, the VTK-m team will ensure that the software compiles, runs, and performs well on the platform.

Ease of Adoption The VTK-m team will address complications with integrating the VTK-m software with other software projects. Part of this work includes isolating external code from device code. Another part includes improving compile times.

4.4.11 WBS 2.3.4.14 VeloC: Very Low Overhead Checkpointing System

Overview The VeloC-SZ project aims to provide VeloC, a high-performance, scalable checkpoint/restart framework that leverages multi-level checkpointing (the combination of several resilience strategies and heterogeneous storage) to ensure that ECP applications run to completion with minimal performance overhead. It delivers a production-ready solution that increases development productivity by reducing the complexity of having to deal with a heterogeneous storage stack and multiple vendor APIs. VeloC offers a client library that can be used by the applications to capture local application states, which are then coordinated and persisted using a resilience engine. VeloC runs the resilience engine asynchronously, which overlaps a large part of the checkpointing with the application runtime, thereby reducing its overhead. An overview of the architecture of VeloC is depicted in Figure 80.

VeloC has been released and shows significant lower checkpointing overhead for several ECP applications, such as HACC, LatticeQCD, EXAALT.

Key Challenges VeloC addresses several key challenges in I/O bottlenecks, deep heterogeneous storage, restart-in-place, and efficient serialization.

I/O Bottlenecks Applications typically employ simple checkpoint-restart mechanisms to survive failures that directly use a parallel file system. With diminishing I/O bandwidth available per core, this leads to high checkpointing overhead and is not sustainable.

Deep Heterogeneous Storage To compensate for diminishing parallel file system I/O bandwidth per core, the storage stack is becoming increasingly deeper and heterogeneous: node-local NVRAM, burst buffers, key-value stores, etc. However, the variety of vendor APIs and performance characteristics make it difficult for application developers to take advantage of it.

Restart-in-Place A majority of failures affect only a small part of the nodes where the job is running. Therefore, reusing the surviving nodes to restart from the latest checkpoint immediately after a failure is more efficient than submitting a new job, which may wait for a long time in the batch queue.

Efficient Serialization The critical data structures of HPC applications that need to be checkpointed are constantly growing in size and complexity. Manual serialization of such data structures as contiguous sequences of bytes to be written into checkpoints is both tedious (leading to loss of productivity) and inefficient (leading to longer runtime). Therefore, automated optimized serialization support is needed.

Solution Strategy To address these challenges, VeloC adopts the following principles.

Multilevel Checkpointing This method is based on the idea that a majority of failures can be mitigated without involving the parallel file system: node-local checkpoints can be used to recover from software bugs, replication/erasure coding can be used to recover from most combinations of node failures. This reduces the frequency of checkpointing to the parallel file system and therefore the I/O bottlenecks.

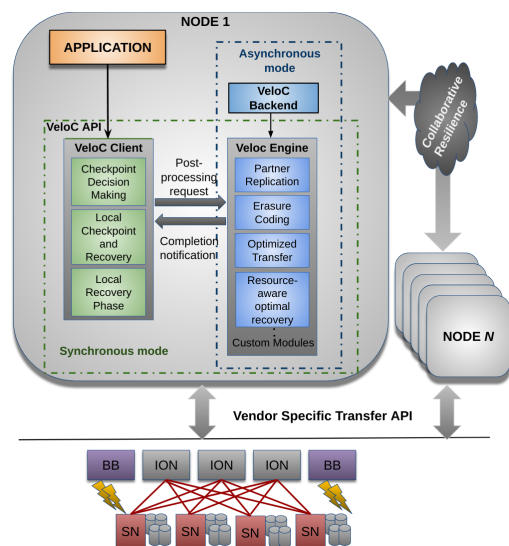


Figure 80: VeloC architecture.

Asynchronous Mode Once a node-local checkpoint has been written, applications do not need to wait for replication, erasure coding or writes to the parallel file system: these can be applied in the background, while the application continues running. However, in this case, it is important to minimize interference with the application execution.

Transparent Use of Heterogeneous Storage We developed several techniques that can leverage a variety of local storage (in-memory file systems, flash storage) and external storage (burst buffers, key-value stores, parallel file systems) options. These techniques select the best available storage options, tune them with the optimal parameters and leverage any vendor-specific API if needed to transfer data.

Job Scheduler Integration To implement restart-in-place, we have developed a series of scripts that interact with a variety of job schedulers to run jobs with spare nodes, continue execution on failures, restart on the surviving nodes and spares using the fastest possible recovery strategy (which ideally avoids reading checkpoints from the parallel file system). This is transparent to the users.

Declarative API and Automated Serialization We offer a simple API that enables users to either manually capture checkpoints into files or to define memory regions that are automatically serialized into checkpoint files.

Modular design Applications link with a client library that is responsible to manage local checkpoints, while a separate engine is responsible to employ the rest of the resilience strategies as pluggable modules. This simplifies the implementation of the asynchronous mode, it enables users the flexibility to choose any combination of resilience strategies, as well as, to customize their checkpointing pipeline (e.g., add new post-processing operations such as analytics or compression).

Recent Progress We met and closely collaborated with several ECP application teams in an effort to address their checkpointing needs. Most of our current efforts involve the HACC, LatticeQCD, and EXAALT teams. We also started integrating VeloC into NWChemEX. Based on feedback from several teams, we extended VeloC to support multiple communication protocols that reduce dependencies on external libraries (such as Boost) if desired. We added preliminary support for checkpoint aggregation, which reduces the number of checkpoint files from one per rank to a desired smaller number.

In terms of capabilities, we refactored the VELOC code to abstract all interactions with external storage in the code, which allows flexible support to persist checkpoints on a variety of options in addition to conventional POSIX-based parallel file systems. In particular, to illustrate this capability, we added support for DAOS, an object-based store developed by Intel and planned to be used by the Aurora supercomputer. This paves the way to include support for other object-based and persistent key-value stores, likely to become popular in the future and developed by the vendors and/or the HPC community.

Second, we continued our efforts to provide an incremental checkpointing capability in VELOC for HPC applications that need to checkpoint data structures that change only partially between the checkpoint requests. Our proposal is exploring hash-based de-duplication to optimize the trade-off between reducing checkpoint sizes at the expense of identifying and consolidating the increments in a heterogeneous environment (GPU + CPU). This has two advantages: (1) it produces smaller checkpoints, which saves storage space across the whole heterogenous storage stack; (2) it improves checkpointing performance by reducing the data transfer overheads, which impact both the blocking phase and the asynchronous operations.

Third, we continued working on the checkpoint aggregation, which allows N local files produced during the blocking phase of VELOC to be aggregated asynchronously into M files on the external storage. Since state of art aggregation approaches (e.g. MPI-IO) are designed for collective blocking I/O, we developed our own algorithms to enable asynchronous aggregation. In particular, we proposed a decoupled peer-to-peer leader election algorithm that allows independent groups of (potentially remote) ranks to consolidate their data on a single rank, which is then responsible to write a single file to the PFS.

Testing, Continuous Integration, and Experience on Early Access Systems We continued working towards new features that facilitate better integration with the ECP ecosystem. We extended the test suite of VELOC (runnable both independently and with continuous integration) to include more comprehensive set of combinations of scenarios, both for VELOC and its subcomponents. In particular, we added support for testing the refactored VELOC code that allows multiple options for external storage and its new DAOS support. Furthermore, we experimented with VELOC at small scale on various early testbeds encouraged for use by the ECP community. Notably, we experimented on Spock, a proxy to Frontier hosted at ORNL. From the VELOC perspective, it features both high performance memories and local storage (NVMe-based SSD), while persistent storage is provided through a PFS mount point. Despite the instability of the software stack on this testbed, we successfully tested VELOC with the standard battery of tests and confirmed its correctness at small scale. Presque (ANL) is another platform we targeted as it provides access to a DAOS deployment. We were successful in running the refactored version of VELOC to checkpoint small data sizes, but encountered issues with DAOS (reported to Intel) when using larger checkpoint sizes.

Next Steps We are working toward several goals to (1) refine incremental checkpointing techniques; (2) refine asynchronous checkpoint aggregation techniques; (3) continue hardening the integration with existing ECP applications; and (4) test VELOC on more ECP testbeds, in particular CRUSHER. In parallel, we will continue the collaboration with the ECP application teams to address new requirements as they arise.

4.4.12 WBS 2.3.4.14 ECP SZ: Fast, Effective, Parallel Error-bounded Exascale Lossy Compression for Scientific Data

Overview Extreme scale simulations and experiments are generating more data than can be stored, communicated and analyzed. Error-bounded lossy compressor is critical because it can get a very high compression ratio while still respecting data fidelity based on user's requirement on compression errors.

The VeloC-SZ project is extending and improving the SZ error-bounded lossy compressor for structured and unstructured scientific datasets. SZ offers an excellent compression ratio as well as very low distortion and compression time. Further work is essential, however, to improve our SZ lossy compressor for ECP

scientific datasets, while ensuring that user-set error controls are respected. Specifically, we are: (i) optimizing SZ compression ratios, accuracy and speed based on end-user needs (ii) refactoring SZ in C++ to support a composable compression framework and all data types used in ECP applications, (iii) integrating SZ in ECP client applications, (iv) developing the GPU version of SZ which supports multiple supercomputers with different architectures (such as Aurora, Frontier and Summit), and (v) improving robustness and testability. We are working with multiple ECP application teams, including ExaSky cosmology teams (HACC), molecular dynamics simulations groups (EXAALT), x-ray laser imaging experimentalists (ExaFEL), and computational chemists (NWChem-X, GAMESS) to optimize SZ for their applications and to harden SZ.

Key Challenges SZ faces several key challenges in improving prediction accuracy, parameter tuning, implementation and optimization on GPU platforms/portability, integrations schemes, and development robustness.

Improving Prediction Accuracy The core step in SZ is the point-wise data prediction for different datasets. The higher the prediction accuracy, the higher the compression quality in general. However, how to design an effective predictor that can adapt to diverse datasets is very challenging.

Parameter Tuning One challenge in optimizing lossy compression for scientific applications is the large diversity of scientific data, dimensions, scales, and dynamic data changes in both space and time. Each application requires specific parameters tuning and in some cases, a specific compression pipeline, which is non-trivial to implement.

Implementation & Optimization for GPUs The third challenge is the sophisticated design in different stages of the SZ (e.g., data prediction, Huffman tree construction, Huffman encoding), which makes the development of efficient GPU kernels non-trivial.

Portable Support for GPUs Optimization of SZ for Aurora and Frontier requires writing portable accelerator codes that are non-trivial for a complex compression pipeline.

Diverse Integration Schemes The fourth challenge is the diversity of the integration schemes for the different ECP client applications: HACC integrates SZ in a proprietary I/O library (GIO), Exafel integrates SZ directly in the LCLS data processing pipeline. GAMESS integrates SZ in the application directly replacing some code sections. EXAALT integrates SZ for saving storage space.

Improve Development Robustness and Testability The SZ testing infrastructure (unit test, correctness test, performance test, regression test, continuous integration) will need to be adapted and its performance optimized for the new C++ implementation. A template based approach must be used to improve robustness, debugging and testability.

Solution Strategy For the first two challenges, we developed a novel, adaptive predictor based on a dynamic spline interpolation, which is combined with the classic Lorenzo predictor, such that the rate distortion (bit rate vs. data distortion PSNR) can be improved significantly. As for the third challenge, we will keep exploring new strategies to improve the GPU kernel performance, especially for the bottleneck stages of the cuSZ-GPU version of SZ. This requires an in-depth understanding of the bottleneck of cuSZ and numerous performance tuning tests to explore new strategies. For the fourth challenge, we refactor SZ in C++, starting from the current C version. This refactoring is the perfect opportunity to implement a new more modular design of SZ that is capable of integrating more stages in the compression pipeline and of selecting compression stages based on specific application data features. Finally, for the fifth challenge, we are in contact with the ALCF, the OLCF, and vendors to access simulators and early systems. We will also develop a portable GPU implementation of SZ for different GPUs by leveraging the corresponding libraries/toolkits such as oneAPI and HIP. Furthermore, we are continuously improving the robustness and testability. We integrated SZ into ORNL's CI environment and often discuss potential issues with application teams when needed.

Recent Progress We summarize 10+ different versions of SZ on our official website,²³ including CPU and GPU versions and a customized version for applications such as EXAALT and GAMESS.

We significantly improved compression quality and performance for SZ. We improved the rate distortion significantly by a dynamic spline interpolation method, which is particularly effective in compressing smooth datasets with wave patterns such as turbulent flow data and quantum chemistry data (QMCPack). We customized an efficient error-bounded lossy compressor called MMD-SZ for molecular dynamics (MD) applications such as LAMMPS and ExaALT by leveraging the potential data characteristics of MD simulation datasets. The compression ratio has been improved by over 100% compared with the generic version of SZ.

We also developed SZ3, which breaks down different stages of SZ to form a loosely coupled model so that the users can construct a new compressor by customizing each compression step conveniently.

Additionally, we significantly improved the GPU kernel performance of SZ’s compression stage. We released the cuda-based SZ (cuSZ) 0.2.9.1 and tested its performance on V100 and A100 GPUs. We identified that the bottleneck of cuSZ is its Huffman encoding stage and dictionary encoding stage. We developed an efficient parallel code book construction for GPUs that scales effectively with the number of input symbols. We proposed a novel reduction-based encoding scheme that can efficiently merge the code words on GPUs. We optimized the overall GPU performance by leveraging the state-of-the-art CUDA APIs such as Cooperative Groups. Finally, we evaluated our Huffman encoder thoroughly using six real-world application datasets on two advanced GPUs and compared the results with our implemented multithreaded Huffman encoder. Experiments show that our solution improves the encoding throughput by up to 5.0× on NVIDIA’s Quadro RTX 5000 and up to 6.8× on NVIDIA’s V100. Figure 81 illustrates the parallel design of the Huffman encoding in cuSZ.

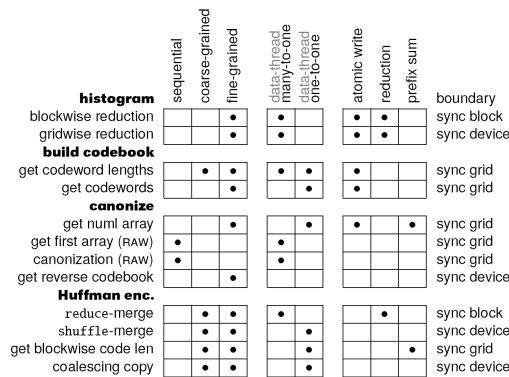


Figure 81: Parallelism implemented for Huffman coding’s subprocedures (kernels): *sequential* denotes that only 1 thread is used due to data dependency, *coarse-grained* denotes that data is explicitly chunked, and *fine-grained* denotes that there is a data-thread mapping with little or no warp divergence.

along with oneAPI SDK serves as the native programming language on A21 system. From DPC++ setup, we successfully executed our prototype kernels that can partially match the native CUDA kernel executed on NVIDIA V100.

We developed CI for SZ on ORNL CI environment and also added smoke test to Spack for SZ.

Preliminary Experiences on Early Access Systems We translated the CUDA-version of SZx (cuSZx) to the HIP-version SZx (hipSZx), then successfully compiled it. To accomplish the translation, we leveraged HIPIFY. There are two tools in HIPIFY: hipify-perl and hipify-clang. We first tried hipify-perl on the Crusher cluster. It is a regular expression-based translator and does not rely on the CUDA toolkits (there is no CUDA module on Crusher). Unfortunately, the hipify-perl translated code yielded many errors during the compilation since this translator only performed simple string replacement without any syntax checks.

²³<http://szcompressor.org>

Hence, we had to use hipify-clang to translate the code again on Summit (it requires a CUDA toolkit). The converted code can be compiled with hipcc on Crusher after moderate adjustments. During the compilation, we observed that the default rocm/4.2.0 does not work with our code; we must load the module rocm/4.5.0 or above versions. With the new rocm version, we had to specify the running platform. For example, we need to define `__HIP_PLATFORM_AMD__` before the main function in hipSZx in order to run it on Crusher. We note that the translated files should be saved in a `.cpp` format. The default `.hip` and the `.c` suffixes do not work as the compiler cannot recognize the GPU-related code snippets in these file formats. There were some known unsupported CUDA instructions in HIP, for instance, the cooperative groups and the warp-level shuffle operations. We had to avoid them for now.

The generated files were executable. However, there were memory access faults that happened during the execution and made the program crash. We had spent some time debugging it and figured out that it was caused by the indexing issue when reading an array in the GPU kernel. These indices were calculated by a GPU-based prefix scan step which frequently used the warp-level shuffle operations. The CUDA shuffles enabled explicit warp synchronization and used masks to allow partial warp shuffling. On the contrary, HIP only implemented the old fashion shuffles that had been deprecated in CUDA. These HIP shuffles did not allow masks and thus needed to involve the entire warp every time. Therefore, the straightforward translation from the CUDA shuffles to the HIP shuffles made the results incorrect in our GPU prefix scan. We must find a way to work around it, e.g., move the prefix scan data from the registers to the shared memory. But this will obviously downgrade the overall performance.

Next Steps Our next efforts include: (1) Improve compression quality and performance for cuSZ, (2) Keep working on the compression quality improvement and integration of SZ in more ECP applications such as ExaFEL and EXAALT, and (3) evaluate the portable GPU version of SZ on ECP platforms.

4.4.13 WBS 2.3.4.15 ExaIO - ExaHDF5

Overview Hierarchical Data Format version 5 (HDF5) is the most popular high-level I/O library for scientific applications to write and read data files. The HDF Group released the first version of HDF5 in 1998 and over the past 20 years, it has been used by numerous applications not only in scientific domains but also in finance, space technologies, and many other business and engineering fields. HDF5 is the most used library for performing parallel I/O on existing HPC systems at the DOE supercomputing facilities. NASA gives HDF5 software the highest technology readiness level (TRL 9), which is given to actual systems flight-proven through successful mission operations. In ECP, numerous applications declared that HDF5 is a critical dependency for performing I/O.

The ExaIO project's ExaHDF5 team, which includes researchers and developers from LBNL, ANL, and The HDF Group, has developed various HDF5 features to address efficiency and other challenges posed by data management and parallel I/O on exascale architectures. The ExaIO-ExaHDF5 team is productizing features and techniques that have been previously prototyped, exploring optimization strategies on upcoming architectures, maintaining and optimizing existing HDF5 features tailored for ECP applications. Along with supporting and optimizing the I/O performance of HDF5 applications, new features in this project include transparent data caching in the multi-level storage hierarchy, topology-aware I/O performance optimization, asynchronous I/O, multi-dataset I/O API, querying data and metadata, and scalable sub-file I/O.

Many of the funded exascale applications and codesign centers require HDF5 for their I/O, and enhancing the HDF5 software to handle the unique challenges of exascale architectures will be instrumental in the success of ECP. For instance, AMReX, the AMR codesign center, is using HDF5 for I/O, and all the ECP applications that are collaborating with AMReX will benefit from improvements to HDF5. The EQSIM project updated its software stack to use HDF5 as a file format for their workflows, achieving significant performance benefits and storage space savings. The ExaIO HDF5 team has worked with numerous ECP AD teams, including E3SM, FLASH, WarpX, ExaSky, etc. and with various DOE supercomputing facilities in achieving superior I/O performance with their HDF5 usage and in integrating enhanced HDF5 features in their codes. The virtual Object Layer (VOL) and interoperability features with PnetCDF and ADIOS data open up the rich HDF5 data management interface to science data stored in other file formats. The dynamically pluggable Virtual File Driver (VFD) feature allows extending HDF5 file format to new sources and destinations of I/O, including GPU memory and cloud storage. The ExaIO HDF5 project has been

releasing these new features in HDF5 for broad deployment on HPC systems. Focusing on the challenges of exascale I/O, technologies will be developed based on the massively parallel storage hierarchies that are being built into pre-exascale systems. The enhanced HDF5 software will achieve efficient parallel I/O on exascale systems in ways that will impact a large number of DOE science as well as industrial applications.

Key Challenges There are challenges in developing I/O strategies for efficiently using a hierarchy of storage devices and topology of compute nodes, developing interoperability features with other file formats, and integrating existing prototyped features into production releases.

Efficient Use of Hierarchical Storage and Topology Data generation (e.g., by simulations) and consumption (such as for analysis) in exascale applications may span various storage and memory tiers, including near-memory NVRAM, SSD-based burst buffers, fast disk, campaign storage, and archival storage. Effective support for caching and prefetching data based on the needs of the application is critical for scalable performance. Also, support for higher bandwidth transfers and lower message latency interconnects in supercomputers is becoming more complex, in terms of both topologies as well as routing policies. I/O libraries need to fully account for this topology in order to maximize I/O performance, and current I/O mechanisms fail to exploit the system topology efficiently.

Asynchronous I/O Asynchronous I/O allows an application to overlap I/O with other operations. When an application properly combines asynchronous I/O with nonblocking communication to overlap those operations with its calculation, it can fully utilize an entire HPC system, leaving few or no system components idle. Thus, adding asynchronous I/O to an application’s existing ability to perform nonblocking communication is necessary to maximize the utilization of valuable exascale computing resources.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each effort.

Asynchronous I/O Virtual Object Layer (VOL) Connector Overlapping the latency incurred in I/O operations with computation or communication operations can hide the I/O latency and improve application performance significantly. Implementation of asynchronous I/O operations can be achieved in different ways. Since the native asynchronous interface offered by most existing operating systems and low-level I/O frameworks (POSIX AIO and MPI-IO) does not include all file operations, I/O operations are executed using a background thread. With the recent increase in the number of available CPU threads per processor, it is now possible to use a thread to execute asynchronous operations from the core that the application is running on without a significant impact on the application’s performance. As shown in Figure 82, when an application enables asynchronous I/O, a background thread is started. Each I/O operation is intercepted, and an asynchronous task is created, storing all the relevant information before inserting it into the asynchronous task queue. The background thread monitors the running state of the application, and only starts executing the accumulated tasks when it detects the application is idle or performing non-I/O operations. When all I/O operations have completed and the application issues the close file call, the asynchronous I/O related resources, as well as the background thread itself, are freed.

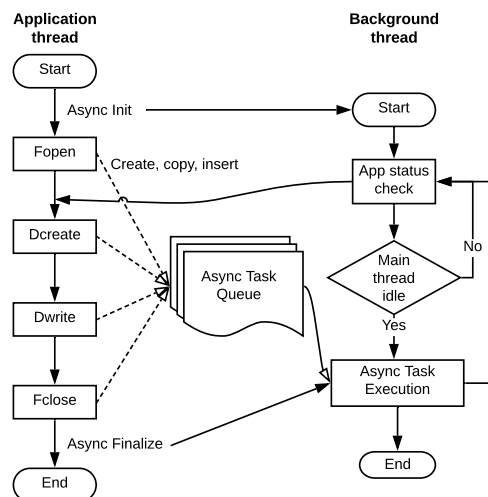


Figure 82: An overview of asynchronous I/O as an HDF5 VOL connector.

Utilizing Complex Compute and Storage Hardware To take advantage of multiple levels of faster storage layers between memory and medium- to long-term storage, the ExaIO HDF5 team has upgraded a previously

developed Data Elevator VOL connector with a new VOL connector called Cache VOL. The Cache VOL intercepts HDF5 file access calls and redirects them to intermediate faster caching storage layers, which future application reads or writes will then access. Data Elevator was extensively tested on burst buffers that were shared by all compute nodes. The team is currently testing it in combination with the asynchronous I/O VOL connector and using a node-local burst buffer layer using UnifyFS.

Recent Progress Recent progress includes developments in VOL framework integration into HDF5, the asynchronous I/O VOL connector, multilevel storage capabilities, and ECP application I/O.

Integration of the VOL framework into the HDF5 develop branch The initial implementation of the VOL feature introduced in the HDF5 1.12.0 release didn't support stack-able VOL connectors. VOL architecture in HDF5 was updated to allow stacking multiple VOL connectors and is now available in the main HDF5 development branch <https://github.com/HDFGroup/hdf5>. A pass-through VOL connector <https://github.com/hpc-io/vol-external-passthrough> also has been developed to test the stack-ability of multiple VOL connectors.

Asynchronous I/O VOL Connector The ExaIO HDF5 team has shown the benefits of using the asynchronous I/O in HDF5 in a TPDS paper (H. Tang, Q. Koziol, S. Byna and J. Ravi, "Transparent Asynchronous Parallel I/O using Background Threads" in IEEE Transactions on Parallel and Distributed Systems, vol. , no. 01, pp. 1-1, 5555. doi: 10.1109/TPDS.2021.3090322). The asynchronous I/O VOL is available for use: <https://github.com/hpc-io/vol-async>.

Developed Methods to Use Multilevel Storage The project team has developed a prototype implementation of using memory and storage layers for caching data. The team previously demonstrated that using the burst buffers on Cori, where the Data Elevator achieves 1.2–3X performance improvement over a highly tuned HDF5 code in reading data. The ExaIO HDF5 team also developed a Cache VOL connector for taking advantage of RAM, node-local cache or any other layer between application buffer and longer-term storage. The current version of the Cache VOL connector is available for testing: <https://github.com/hpc-io/vol-cache>.

Supporting ECP Application I/O The ExaIO HDF5 team has been working with various applications in the ECP portfolio. Applications in the AMReX codesign center have seen some performance issues, mainly because of less optimal configurations, such as using too few file system servers (e.g. Lustre Object Storage Targets or OSTs), producing a large number of metadata requests, using MPI collective buffering that was observing poor performance on NERSC's Cori. By simply changing these configurations, HDF5 achieved higher performance in writing files. The team also tuned HDF5's I/O performance by more than 10X by setting the alignment parameter that matches the block size of the GPFS file system on Summit at the OLCF. The team extended supporting ECP applications, such as E3SM, WarpX/OpenPMD, and FLASH codes by identifying I/O performance bottlenecks and mitigating them. The team has shown improving I/O performance for I/O benchmarks representative of these applications by up to 8X and is in the process of integrating them into the application codes.

The ExaIO HDF5 team has been building and testing the new HDF5 features on various EAS systems including Perlmutter, Spock, Crusher, and Presque. The asynchronous I/O and cache VOL features, and the h5bench were successful in building. All small-scale tests passed with the develop branch of HDF5.

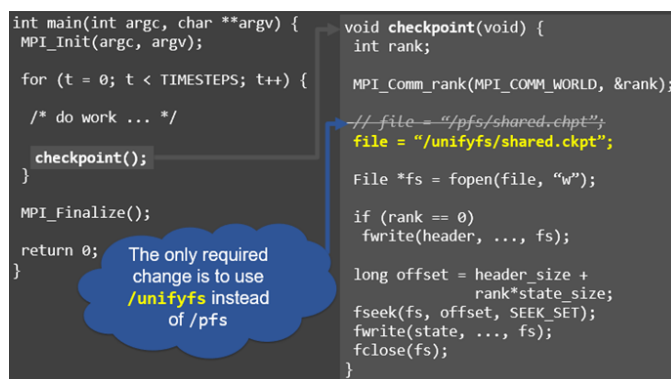
Next Steps The ExaIO HDF5 team is developing subfiling for reducing locking and contention on parallel file systems, fine-tuning asynchronous I/O and the Cache VOL connectors for caching and prefetching, and supporting ECP AD and ST teams and facilities in improving the overall performance of HDF5. The team is also working on tuning parallel compression and developing multi-dataset I/O API in HDF5. The team will continue testing and measuring performance benefits on EAS systems.

4.4.14 WBS 2.3.4.15 UnifyFS – A File System for Burst Buffers

Overview The view of storage systems for HPC is changing rapidly. Traditional, single-target parallel file systems have reached their cost-effective scaling limit. As a result, hierarchical storage systems are being designed and installed for our nation's next-generation leadership class systems. Current designs employ

“burst buffers” as a fast cache between compute nodes and the parallel file system for data needed by running jobs and workflows. Burst buffers are implemented as compute-node local storage (e.g., SSD) or as shared intermediate storage (e.g., SSD on shared burst buffer nodes).

Because burst buffers present an additional complexity to effectively using supercomputers, we developed UnifyFS, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. UnifyFS addresses a major usability factor of current and future systems, because it enables applications to gain the performance advantages from distributed burst buffers while providing ease of use similar to that of a parallel file system. To use UnifyFS from within an MPI application, one only needs to change the paths for files that the application uses from the parallel file system to the mount point for UnifyFS (Figure 83). Then the application performs I/O as it normally would, using POSIX I/O or a high level I/O library, e.g., HDF5 or MPI-IO. The UnifyFS library intercepts all I/O operations and manages the file data locally on the compute nodes with high performance.



```

int main(int argc, char **argv) {
    MPI_Init(argc, argv);

    for (t = 0; t < Timesteps; t++) {
        /* do work ... */
        checkpoint();
    }

    MPI_Finalize();
    return 0;
}

void checkpoint(void) {
    int rank;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    // file = "/pfs/shared.chkpt";
    file = "/unifyfs/shared.ckpt";

    File *fs = fopen(file, "w");

    if (rank == 0)
        fwrite(header, ..., fs);

    long offset = header_size +
        rank*state_size;
    fseek(fs, offset, SEEK_SET);
    fwrite(state, ..., fs);
    fclose(fs);
}

```

The only required change is to use /unifyfs instead of /pfs

Figure 83: Using UnifyFS from an MPI application is as easy as using the parallel file system. Simply change the file path to point to the UnifyFS mount point /unifyfs and then perform I/O as normal.

Key Challenges The hierarchical storage of current and future HPC systems includes compute-node local SSDs as burst buffers. This distributed burst buffer design promises fast, scalable I/O performance because burst buffer bandwidth and capacity will automatically scale with the compute resources used by jobs and workflows. However, a major concern for this distributed design is how to present the disjoint storage devices as a single storage location to applications that use shared files. The primary issue is that when concurrent processes on different compute nodes perform I/O operations, e.g., writes, to a shared file, the data for the file are scattered across the separate compute-node local burst buffers instead of being stored in a single location. Consequently, if a process wants to access bytes from the shared file that exist in the burst buffer of a different compute node, that process needs to somehow track or look up the information for locating and retrieving those bytes. Additionally, there is no common interface across vendors for accessing remote burst buffers, so code for cross-node file sharing will not be easily portable across multiple DOE systems with different burst buffer architectures, further increasing programming complexity to support shared files.

For the reasons outlined above, it is clear that without software support for distributed burst buffers, applications will have major difficulties utilizing these resources.

Solution Strategy To address this concern, we have developed UnifyFS, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. In Figure 84, we show a high level schematic of how UnifyFS works. Users load UnifyFS into their jobs from their batch scripts. Once UnifyFS is instantiated, user applications can read and write shared files to the mount point just like they would the parallel file system. File operations to the UnifyFS mount point will be intercepted and handled with specialized optimizations that will deliver high I/O performance.

Because bulk-synchronous I/O dominates the I/O traffic most HPC systems, we target our approach at those workloads. Examples of bulk-synchronous I/O include checkpoint/restart and periodic output

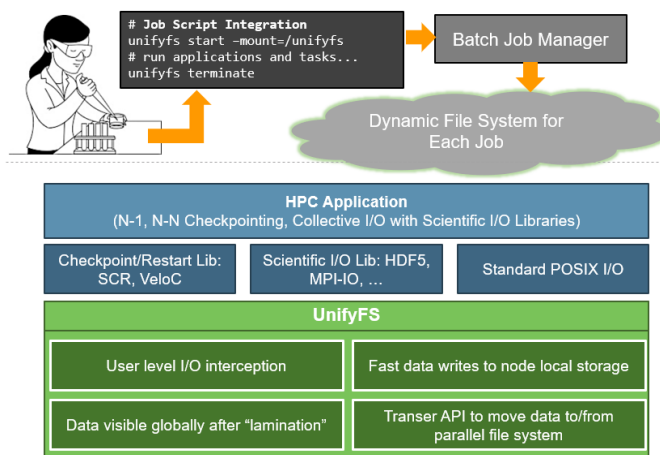


Figure 84: UnifyFS Overview. Users can give commands in their batch scripts to launch UnifyFS within their allocation. UnifyFS works transparently with POSIX I/O, common I/O libraries, and VeloC. Once file operations are intercepted by UnifyFS, they are handled with specialized optimizations to ensure high performance.

dumps by applications. Thus, UnifyFS addresses a major usability factor of current and future systems. We designed UnifyFS such that it transparently intercepts I/O calls, so it will integrate cleanly with other software including I/O and checkpoint/restart libraries. Additionally, because UnifyFS is tailored for HPC systems and workloads, it can deliver high performance.

Recent Progress In the past year, the UnifyFS team has focused on improving the usability and performance of UnifyFS for applications and I/O middleware. We evaluated UnifyFS for use by producer-consumer applications, which include applications such as climate models that exchange data files between independent components such as ocean and atmosphere, and applications that couple traditional HPC simulation with ML analysis. We updated UnifyFS to better support these types of applications, including documentation additions and fixes to the UnifyFS code base. This year, we also introduced our UnifyFS API which is intended to be used by I/O middleware software, e.g., I/O libraries like HDF5. By using this API, I/O middleware can have finer control over the workings of UnifyFS and obtain better performance. Finally, we also implemented numerous changes and bug fixes in preparation for our support of UnifyFS on Frontier. Our source code for UnifyFS is available on GitHub at <https://github.com/LLNL/UnifyFS>. Within the last year, we have also deployed UnifyFS on Summit as a production ready module.

The UnifyFS team has also made progress on the EAS systems for Frontier. As of this writing, the team has successfully built and run UnifyFS at small scale on Spock. The tests completed without errors and exhibited expected performance. We have begun testing on Crusher and will continue that effort.

Next Steps Our efforts for the next year will be focused on delivering a robust and high-performance implementation of UnifyFS on Frontier. We have begun the work of porting to early access systems for Frontier and will continue on this track. We will rigorously test the correctness and performance of UnifyFS and will evaluate UnifyFS's performance with ECP applications on Frontier.

4.4.15 WBS 2.3.4.16 ALPINE

Overview ECP ALPINE/ZFP will deliver in situ visualization and analysis infrastructure and algorithms along with lossy compression for floating point arrays to ECP Applications.

ALPINE infrastructure developers come from the ParaView [215, 216] and VisIt [217] teams and ALPINE solutions will deliver in situ DAV functionality in those tools, as well as through Ascent [218], a new in situ infrastructure framework that focuses on flyweight processing. ALPINE focuses on four major activities:

1. Deliver Exascale visualization and analysis algorithms that will be critical for ECP Applications as the dominant analysis paradigm shifts from post hoc (post-processing) to in situ (processing data in a code as it is generated).
2. Deliver an Exascale-capable infrastructure for the development of in situ algorithms and deployment into existing applications, libraries, and tools.
3. Engage with ECP Applications to integrate our algorithms and infrastructure into their software.
4. Engage with ECP Software Technologies to integrate their Exascale software into our infrastructure.

Key Challenges Many high performance simulation codes are using post hoc processing. Given Exascale I/O and storage constraints, in situ processing will be necessary. In situ data analysis and visualization selects, analyzes, reduces, and generates extracts from scientific simulation results during the simulation runs to overcome bandwidth and storage bottlenecks associated with writing out full simulation results to disk. The ALPINE team is addressing two problems related to Exascale processing: (1) delivering infrastructure and (2) delivering high-performance, in-situ algorithms. The challenge for existing infrastructure tools is that need to become Exascale-ready in order to achieve performance within simulation codes' time budgets, support many-core architectures, scale to massive concurrency, and leverage deep memory hierarchies. The challenge for in situ algorithms is to apply in situ processing effectively without a human being in the loop. This means that we must have adaptive approaches to automate saving the correct visualizations and data extracts.

Solution Strategy A major strategy for our team is to leverage existing, successful software, ParaView and its in situ Catalyst [219] library and VisIt, and then to integrate and augment them with ALPINE data and analysis capabilities to address the challenges of Exascale. Both software projects represent long-term DOE investments, and they are the two dominant software packages for large-scale visualization and analysis within the DOE SC and the DOE NNSA. Our team is also developing an additional in situ framework, Ascent. Ascent is a flyweight solution, meaning that it is focused on a streamlined API, minimal memory footprint, and small binary size. Our solution strategy is two-fold, in response to our two major challenges: infrastructure and algorithms.

For infrastructure, we have developed a layer on top of the VTK-m library for ALPINE algorithms. This layer is where all ALPINE algorithms will be implemented, and it is deployed in ParaView/Catalyst, VisIt, and Ascent. Thus all development effort by ALPINE will be available in all of our tools and by leveraging VTK-m, we will be addressing issues with many-core architectures.

Furthermore, ALPINE is developing a suite of in-situ algorithms designed to address I/O and data output constraints and enable scientific discovery:

Topological Analysis Used to detect features in the data and adaptively steer visualizations with no human in the loop. For example, contour trees can identify the most significant isosurfaces in complex simulations and then the resulting visualizations can use these isosurfaces [220].

Adaptive Sampling This can be used to guide visualizations and extracts to the most important parts of the simulation, significantly reducing I/O [221, 222, 223].

Statistical Feature Detection These models use distribution-based approaches and statistical similarity measures to identify and isolate features of interest [224, 225]. Significant data reduction is possible by only saving the statistical representations of the data. Figure 85 illustrates the use of the statistical feature detection approach to identifying bubbles in situ in an MFIX-Exa simulation.

Task-based Feature Extraction This method uses segmented merge trees to encode a wide range of threshold based features. An embedded domain specific language (EDSL) can describe algorithms using a task graph abstraction [226, 227] and execute it using different runtimes (e.g., MPI, Charm++, Legion).

Optimal Viewpoint Optimal Viewpoint metrics can be used to automate visualization decisions while running in situ. The initial algorithm implementation will choose the best camera placement for a scene, minimizing visualizations written to disk [228, 229].

Lagrangian Analysis Using this method to analyze vector flow allows more efficient and complete tracking of flow. It can save vector field data with higher accuracy and less storage than the traditional approaches [230, 231, 232].

Recent Progress During this past year, ALPINE focused on porting infrastructure to early access systems; hardening algorithms by porting to VTK-m and ensuring comprehensive unit testing while furthering integrations with key clients. ALPINE facility team members worked closely with the VTK-m team on porting activities such as builds on early access systems (EAS) and HIP implementations.

The ParaView/Catalyst team updated the build system to enable VTK-m with the CUDA back-end and tested on OLCF's Summit. Likewise, the VisIt team deployed on Summit. A build of ParaView with minimal dependencies was also done on Spock. Deployments included improvements to Spack build processes. Team efforts also covered build and continuous integration activities for ALPINE infrastructure on NDA-restricted early access systems. New VTK-m functionality such as distributed memory particle advection was deployed in Ascent. Efforts are currently underway to deploy ALPINE infrastructure on Perlmutter.

Algorithms have focused on hardening, deployment in VTK-m, and unit testing while integration with clients has continued. With the sampling algorithm now fully GPU capable and the Ascent <> ExaSky:Nyx integration in place, the sampling algorithm team has focused on post hoc reconstruction methods and support for AMR levels.

The Ascent <> ExaLearn <> Pele integration has made considerable progress by developing a full integration of ExaLearn's Genten, a Kokkos-based library for anomaly detection, Ascent and PeleLM. This include updating the co-kurtosis filter in Ascent to directly access the Genten library. In addition, the teams have maintained the PeleC pipeline with updated codes. The joint effort is also exploring interesting science use cases for CombustionPele.

ALPINE and MFIX-Exa teams are collaborating on bubble finding with the statistical feature detection algorithm and a Catalyst integration. The statistical feature algorithm has been deployed in VTK-m, verified with the Catalyst <> MFIX-Exa pipeline and has been tested with over 54 million particles per time step, running on NERSC's Cori. Additional work has been done to support bubble tracking and data summarization techniques for further data reduction.

The contour tree algorithm was refactored to update to latest VTK-m functionality. The team developed an OpenMP+MPI prototype to compute derived metrics based on a distributed contour tree representation. These importance metrics are needed to identify important features found by this topological approach. Additional regression and consistency tests were added as the algorithm has been hardened.

ALPINE is developing a task-based feature detection approach that can be executed on non-MPI runtimes such as Charm++. The task-based feature detection team made significant progress this year, developing the LegoFlow extension which enables composable patterns, making it easier to construct larger task graphs. They enabled a LegoFlow-based pipeline for image compositing as well as parallel merge tree computation, testing with PeleLM and developed a workflow for relevance field computation.

The optimal viewpoint team also made significant strides, parallelizing the algorithm, exploring optimal viewpoint metrics, developing search methods, and studying the temporal characteristics of optimal viewpoint metrics. They ran a user evaluation study to identify the metrics most correlated with human preference. An entropy-based metric was found to be most highly correlated and was used in the development and optimization of the optimal viewpoint over time (OVPOT) algorithm.

The ALPINE/ZFP team has been active in porting and testing its infrastructure, algorithms, and compression capabilities to Spock and Crusher. ALPINE's visualization applications, ParaView and VisIt, have been deployed on Spock CPUs using OSMesa for rendering. Client/server and batch mode were successfully tested. Additionally, ParaView has a preliminary build with VTK-m's Kokkos/HIP backend running on the Spock GPU.

ALPINE's algorithm teams have been testing stand-alone versions of the contour tree algorithm and the statistical feature detection algorithm. On both Spock and Crusher, the contour tree algorithm team leveraged VTK-m's Kokkos/HIP backend to build and run on the CPU for a single node, serial mode test.

On Spock CPUs, the statistical feature detection algorithm team successfully ran using VTK-m's OpenMP backend. We measured a 25x performance improvement from one to 64 threads using OpenMP acceleration.

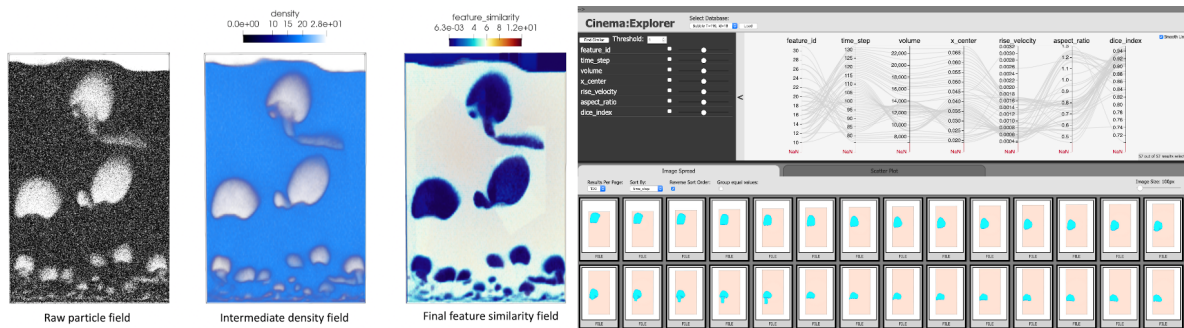


Figure 85: The ALPINE statistical feature detection algorithm is used to identify bubbles in situ in an MFIX-Exa fluidized bed simulation. The raw article data is converted to a particle density field. A threshold is applied to the density field to create a feature similarity field, separating the bubbles from uninteresting regions. Saving only the statistical representation allows greater temporal resolution while significantly reducing output data size. A preliminary study shows a factor of 300 reduction in data size compared to the raw particle fields. The statistical bubble representation becomes the input to a post hoc Cinema-based workflow to track bubbles and explore bubble dynamics.

Next Steps Plans for FY 22–23 will continue the focus on integration and delivery to ECP applications. This will include deploying key infrastructure and integration pipelines on Frontier. The team will also continue outreach with ECP application codes to support ECP applications codes as they shift their focus from KPP-1 or KPP-2 runs to science runs and visualization needs.

4.4.16 WBS 2.3.4.16 ZFP: Compressed Floating-Point Arrays

Overview One of the primary challenges for exascale computing is overcoming the performance cost of data movement. Far more data is being generated than can reasonably be stored to disk and later analyzed without some form of data reduction. Moreover, with deepening memory hierarchies and dwindling per-core memory bandwidth due to increasing parallelism, even on-node data motion between RAM and registers makes for a significant performance bottleneck and primary source of power consumption.

ZFP is a floating-point array primitive that mitigates this problem using very high-speed, lossy (but optionally error-bounded) compression to significantly reduce data volumes. ZFP reduces I/O time and off-line storage requirements by 1–2 orders of magnitude depending on accuracy requirements, as dictated by user-set error tolerances. Unique among data compressors, ZFP also supports constant-time read/write random access to individual array elements from compressed storage. ZFP’s compressed arrays can often replace conventional arrays in existing applications with minimal code changes. This allows the user to store tables of floating-point data in compressed form that otherwise would not fit in memory, either using a desired memory footprint or a prescribed level of accuracy. When used in numerical computations to represent evolving state, ZFP arrays provide a fine-grained knob on precision while achieving accuracy comparable to IEEE floating point at half the storage, reducing both memory usage and bandwidth.

This project is extending ZFP to make it more readily usable in an exascale computing setting by parallelizing it on both the CPU and GPU, targeting several back-ends (OpenMP, CUDA, HIP, SYCL); by providing bindings for several programming languages (C, C++, Fortran, Python); by adding new functionality, e.g., for unstructured data and spatially adaptive compressed arrays; by hardening the software and adopting best practices for software development; and by integrating ZFP with a variety of ECP applications, I/O libraries, and visualization and data analysis tools.

Key Challenges There are several challenges to overcome on this project with respect to implementing compressed floating-point arrays:

Data Dependencies Compression by its very nature removes redundancies, often by deriving information from what has already been (de)compressed and learned about the data. Such data dependencies can usually be resolved only by traversing the data in sequence, thus complicating random access and parallelism.

Random Access For inline compression, on-demand random access to localized pieces of data is essential. However, compression usually represents large fixed-length records using variable-length storage, which complicates random access and indexing.

Parallelism Manycore architectures allow for massively concurrent execution over millions or billions of array elements. Yet compression is usually a process of reducing such multidimensional arrays to a single-dimensional sequence of bits, which requires considerable coordination among parallel threads of execution.

Unstructured Data Unstructured data, such as independent particles and arbitrarily connected nodes in a mesh, has no natural ordering, repeated structure, or regular geometry that can be exploited for compression.

Performance For inline compression to be useful, both compression and decompression have to be extremely fast (simple), yet effective enough to warrant compression. Moreover, the complexities of compression must be hidden from the user to promote adoption, while allowing sufficient flexibility to support essentially arbitrary data access patterns.

These challenges often suggest conflicting solutions and are further complicated by the extreme demands of exascale computing applications.

Solution Strategy ZFP is unique in supporting read and write random access to multidimensional data, and was designed from the outset to address some of the above challenges. The following strategies are employed on this project to overcome the remaining challenges:

Partitioning The d -dimensional arrays are partitioned into small, independent blocks of 4^d scalars each. This enables both fine-grained random access and a large degree of data parallelism.

Fixed-Size Storage Instead of storing fixed-precision values using variable-size storage, ZFP uses fixed-size storage to represent values at the greatest precision afforded by a limited bit budget.

Adaptive Storage For applications that demand error tolerances, this project is developing adaptive representations that allocate bits to where they are most needed, which involves efficient management of variable-length records that might expand and shrink in size over time.

Parallelism OpenMP, CUDA, and HIP implementations of ZFP have been developed that exploit fine-grained data parallelism. Opportunities for task parallelism have also been identified.

Preconditioning The irregularity and unpredictability of unstructured data is improved using *preconditioners* that massage the data to make it more amenable to compression by ZFP. Strategies include sorting, binning, structure inference, transposition, pre-transforms like wavelets, etc.

Abstraction Concrete details about compression, caching, parallelism, thread safety, etc., are abstracted away from the user by providing high-level primitives that make ZFP arrays appear like uncompressed arrays, in part via C++ operator overloading. We are designing classes and concepts commonly available for uncompressed arrays, such as proxy references and pointers into compressed storage that act like their uncompressed counterparts; views into and slices of arrays; and iterators compatible with STL algorithms. Such primitives make it easier to write generic code for which ZFP arrays may easily be substituted for uncompressed arrays.

Recent Progress We completed implementation and refactoring of ZFP's new read-only array classes. These support random access to variable-length blocks through data structures that compactly encode block offsets within the compressed stream. We significantly extended the C wrappers around ZFP's C++ array classes to support 4D arrays, views into arrays, and random-access iterators. With help from NVIDIA, we developed a CUDA implementation of parallel variable-rate compression, which achieves up to 500 GB/s throughput (Figure 86). We further ported our CUDA implementation to HIP and are working with

external collaborators on a SYCL port. We also expanded ZFP testing to more diverse platforms and compilers, including NVIDIA and AMD GPUs, through the adoption of GitLab CI. Finally, through ongoing collaborations with the University of Utah, we published a new point set compression technique [233]. In addition to ST integrations with ADIOS, HDF5, STRUMPACK, and VTK-M, we are integrating ZFP with CEED, EQSIM (Figure 87), QMCPACK/RMG, and WARPX.

The results of our R&D efforts have been documented through publications [234, 235, 236, 237, 238, 239, 233], and significant efforts have been made to reach out to customers and the HPC community at large through one-on-one interactions and tutorials, both at ECP meetings and conferences [240, 241, 242, 243, 244, 245, 246, 247]. Together with the SZ team, we will be giving a tutorial at SC21 [248].

ZFP has been ported to HIP and is successfully running on Spock GPUs. The ZFP team is focusing on performance optimization, in collaboration with the ALPINE and VTK-m teams.

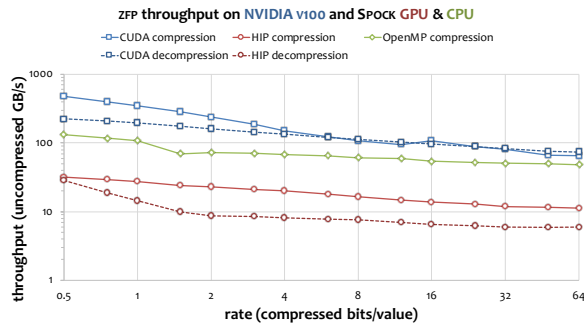


Figure 86: ZFP (de)compression performance.

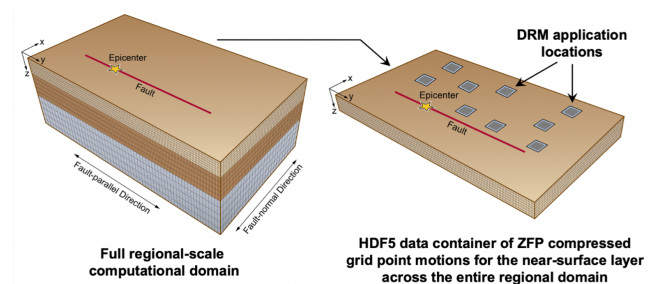


Figure 87: 250:1 ZFP compression of EQSIM seismic wave velocity data (figure courtesy of McCallen et al. [8]).

Next Steps Next year’s effort will focus on completing support for parallel decompression of variable-rate streams, performance optimization, and filling functionality gaps across back ends and languages. We will also prioritize integration efforts with ECP applications and software technologies.

4.5 WBS 2.3.5 SOFTWARE ECOSYSTEM & DELIVERY

End State: A production-ready software stack delivered to our facilities, vendor partners, and the open-source HPC community.

4.5.1 Scope and Requirements

The focus of this effort is on the last mile delivery of software that is intended to be supported by DOE Facilities and/or vendor offerings. The scope of this effort breaks down into the following key areas:

- Hardening and broad ST and facility adoption of Spack for easy build of software on all target platforms.
- Delivery of formal software releases (E4S) in multiple packaging formats technologies – from-source builds, modules, and containers.
- Oversight of the ST SDKs developed in all five ST L3 areas, with a goal of ensuring the SDKs are deployed as production-quality products at the Facilities, and available to the broader open-source HPC community through coordinated releases.
- Development of testing infrastructure (e.g., Continuous Integration) in collaboration with HI 2.4.4 (Software Deployment at Facilities) for use by ECP teams at the Facilities.
- Development and hardening of methods for software deployment through the use of container technology.
- Development and hardening of a scientific workflows SDK that provides a robust, well-tested, well-documented, and scalable set of tools and components that can be used to produce scalable and portable workflows for a wide range of applications.

- Informal partnerships with the Linux Foundation’s OpenHPC project for potential broader deployment of ST technologies in the OpenHPC ecosystem.

A major goal of ST is to ensure that applications can trust that ST products will be available on DOE Exascale systems in a production-quality state, which implies robust testing, documentation, and a clear line of support for each product. This will largely be an integration effort building on both the SDKs project elements defined in each ST L3 area, and tight collaboration and coordination with the HI L3 area for Deployment of Software at Facilities (WBS 2.4.4). We will work to develop, prototype, and deliver foundational infrastructure for technologies such as continuous integration and containers in tight collaboration with our DOE facility partners. The ultimate goal is ensuring that the ECP software stack is robustly supported, as well as finding a reach into the broader HPC open-source community—both of which provide the basis for long-term sustainability required by applications, software, Facilities, and vendors who rely upon these products.

Spack has broad adoption in the open-source community as an elegant solution toward solving many of the challenges presented by building software with many dependencies. Spack is one of the most visible outward-facing products in this L3 area, and is the basis for the SDK and E4S efforts.

4.5.2 Assumptions and Feasibility

Success in this effort will require a coordinated effort across the entire hardware and software stack — in particular with HI 2.4.4 (Delivery of Software at Facilities) and in some cases, our vendor partners. This cooperation is a critical first step in enabling our goals, and this area will drive toward ensuring those partnerships can flourish for mutual gain.

Given the project timelines and requirements of production systems at our Facilities, we do not envision a wholly new software stack as a feasible solution. We do however recognize that in many cases the software of today’s HPC environments will very likely need to either be evolved or extended to meet the mission goals. This will require first, proof-of-concept on existing pre-Exascale hardware, and ultimately adoption of technologies by system vendors where required, and by other application and software teams.

4.5.3 Objectives

This area will focus on all aspects of integration of the ECP software stack embodied in E4S and the development of SDK Community Policies, and building the workflows community and deploying a scientific workflows SDK that provides a robust, well-tested, well-documented, and scalable set of tools and components that can be used to produce scalable and portable workflows for a wide range of applications, with a focus on putting the infrastructure in place (in partnership with HI and the SDKs) for production-quality software delivery through technologies such as Spack, continuous integration, and containers.

4.5.4 Plan

Version 21.08 of the E4S was released in August 2021, which includes 88 ST products that have Spack packages. This release is downloadable from DockerHub under the `ecpe4s` area. The E4S Spack Build Cache now includes binaries for `ppc64le` as well as `x86_64` and includes over 53,000 total binaries. The E4S containers now support custom images for ECP applications, such as `WDMApp`, `ExaWind`, and `Pantheon`. A regular cadence of E4S releases will continue, with updated ST products, broader facility adoption, and potentially inclusion in vendor offerings.

In close coordination with E4S, a number of SDKs are being developed across the other L3 ST areas, building on the years of experience the `xSDK` (Math Libraries). These SDKs will become a prime vehicle for our delivery strategy, while also providing ST products with a standard set of community policies aimed at robust production-ready software delivery. In October 2020, Version 1 of the E4S Community Policies was announced. The E4S Community Policies will serve as membership criteria for E4S member packages. The E4S Community Policies will continue to evolve as we work toward release of Version 2 of them.

Spack continues to gain penetration across the ECP, and will be the de facto delivery method for ST products building from source. We provide Spack packaging assistance for ST users and DOE Facilities, and are developing new capabilities for Spack that enable automated deployments of software at Facilities,

in containerized environments, and as part of continuous integration. Running test suites within Spack environments via the `spack test` functionality is now available. Concurrently, we are developing technologies and best practices that enable containers to be used effectively at Facilities, and are pushing to accelerate container adoption within ECP.

As scientific workflows continue to become more complex, the ExaWorks project is addressing this complexity by co-designing an open Software Development Kit consisting of workflow management systems (WMSs) that can be composed and interoperate through common interfaces. The project is working to bootstrap the SDK with an initial set of tools, design common interfaces, make tools easier to apply to complex science challenges, and apply the SDK to ECP applications. Importantly, the project will not create a new workflow system nor does it aim not to replace the many workflow solutions already deployed and used by scientists, but rather it will provide a well-engineered and scalable SDK which can be leveraged by new and existing workflows. The SDK will be available via E4S and applied to ECP application to address broad workflow needs.

4.5.5 Risks and Mitigation Strategies

Risks are as follows:

- Deploying E4S on unknown architectures; use Spack for deployment to decrease installation complexity
- Keep updated versions of ST and dependent software in sync after initially achieving SDK interoperability
- Delays in deploying a common CI infrastructure lead to subsequent delays in an integrated software release
- Multiple container technologies in flight will make it difficult to reach consensus on a common solution; may not be possible to generate containers that are both portable and perform well
- ECP Application Reliance on workflow management systems that may not be scalable or perform well at exascale; mitigate by adopting robust, well-tested, and scalable components
- OpenHPC partnership is ill-defined and unfunded
- Sustainability of ECP ST capabilities after ECP has ended

4.5.6 Future Trends

SDKs will gain further traction in their communities as the benefits of interoperability and community policies are demonstrated. We believe these processes will become embedded into the communities and become one of the lasting legacies of ECP and critical for sustainability beyond ECP.

Software deployments will continue to become more complex, especially when we require optimized builds for the unique and complicated exascale architectures. Keeping dependencies updated and the software tested on these systems using continuous integration will tax the resources at the Facilities. Software testing that includes interoperability and scalability tests will require further resources, both in terms of people to write the tests and the hours to regularly run them. These put greater emphasis on using and updating Spack as a solution strategy for large collections of software and tight coordination with HI and Facilities on CI infrastructure and resources.

Containers will become more popular and usable as a way to package the entire environment necessary to run an application on the exascale machines, thereby managing some of the complexity of an application deployment. We expect that performance of an application within a container will be nearly as fast as running the application on bare metal. Application build time will be reduced by using the associated build caches.

Workflows will continue to become more complex to complete their science missions, requiring orchestration of many applications and scripts, executed at various scales across many different resource types, and often reliant on machine learning algorithms for guidance. We expect that hardening workflow management systems and building a community centered around robust and scalable components will be foundational for addressing these complexities. Moreover, we expect that container-based scientific workflows will begin to take off as we transition from demonstrations of applications at scale to performing science with them.

4.5.7 WBS 2.3.5.01 Software Development Kits

Overview The ST SDK project supports a set of activities focused on the following:

- Establishing Community Policies aimed at increasing the interoperability between and sustainability of ST software packages, using the xSDK [154] community package and installation policies [3] as a model.
- Coordinating the delivery of ECP ST products through the E4S [249], a comprehensive and coherent set of software tools, to all interested stakeholders on behalf of ECP ST, including ECP applications and the broader open source community.

An ECP ST SDK is a collection of related software products (called packages) where coordination across package teams will improve usability and practices and foster community growth among teams that develop similar and complementary capabilities. SDKs have the following attributes:

- Domain scope: Collection makes functional sense.
- Interaction model: How packages interact; compatible, complementary, interoperable.
- Community policies: Value statements; serve as criteria for membership.
- Community interaction: Communication between teams; bridge culture; common vocabulary.
- Meta-infrastructure: Encapsulates; invokes build of all packages (Spack); shared test suites.
- Coordinated plans: Inter-package planning – does not replace autonomous package planning.
- Community outreach: Coordinated, combined tutorials; documentation; best practices.

The SDK project is needed within ECP because it will make it simpler for ECP applications to access required software dependencies on ECP target platforms and drastically lower the cost of exploring the use of additional ECP ST software that may be of benefit. In addition, the SDK effort will decrease the ECP software support burden at the major computing facilities by ensuring the general compatibility of ST packages within a single software environment, providing tool support for the installation of ST packages on facility machines, communicating common requirements for ST software and facilitating the set up of CI testing at the Facilities. This project works closely with the HI 2.4.4 Deployment of Software at the Facilities project.

Key Challenges ST software packages have been developed in a variety of very different cultures and are at significantly different levels of software engineering maturity and sophistication. The experience of some of the SDK staff during the formation of the xSDK showed that in this situation, it is challenging to establish common terminology and effective communication, and these are prerequisites to community policies and a robust software release.

Deciding exactly how to deploy the SDKs at the Facilities is itself a challenge. ECP applications will use different combinations of ST software in different configurations. For example, applications will want mathematical libraries capabilities from the xSDK build on top of both MPICH and OpenMPI, and will want different configurations of those mathematical libraries.

Solution Strategy The SDK solution strategy involves pursuing interoperability and sustainability goals by grouping ST software projects into logical collections whose members will benefit from a common set of community policies as well as increased communication between members to standardize approaches where sensible and establish better software practices.

The SDK effort will also facilitate the use of common infrastructure, such as CI testing at the major computing Facilities and the Spack [1] package manager. SDK release and delivery goals will benefit from common package manager and testing infrastructure, including the E4S initiative to provide prebuilt binaries for a variety of architectures.

Recognizing the release readiness and broader maturity differences between ECP ST products, the release strategy has been to include only those products ready for a joint release in the E4S releases, but to also continue to work with other products in preparing for subsequent release opportunities.

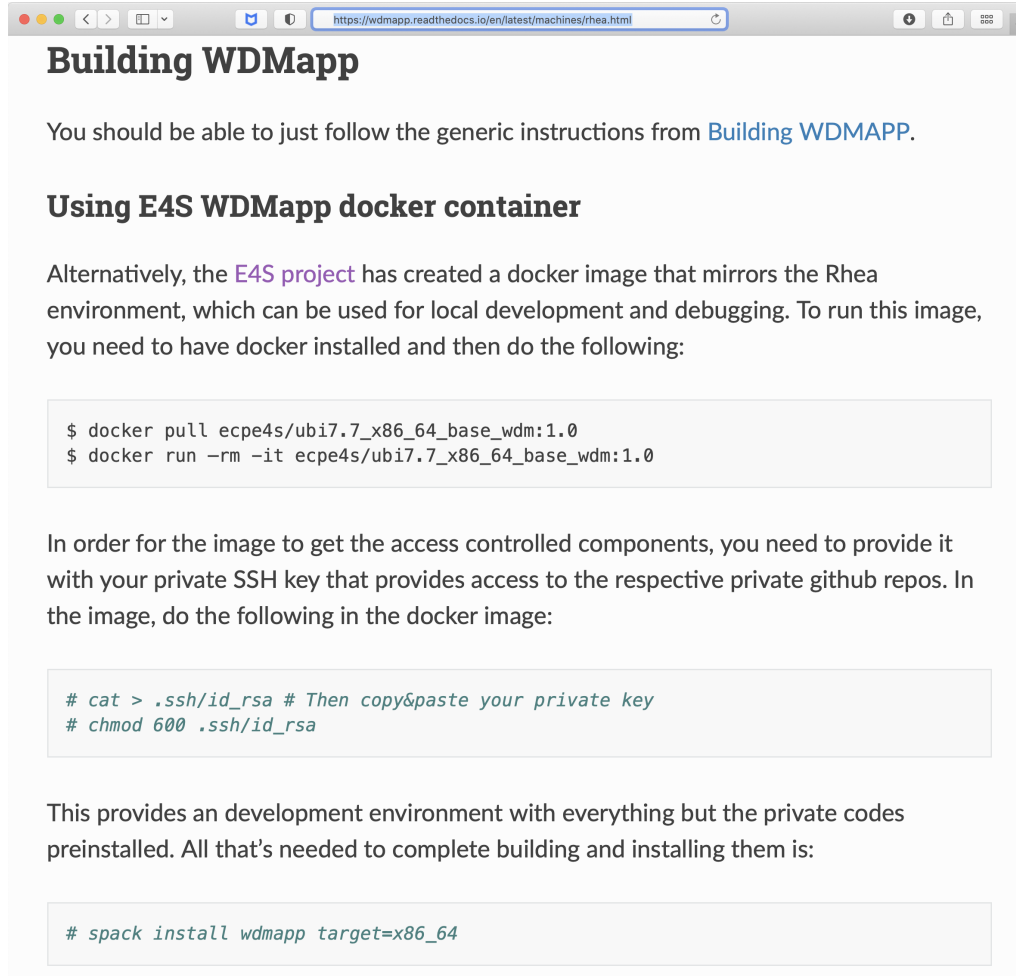


Figure 88: WDMapp documentation for how to use the E4S WDMapp Docker container to speed up WDMapp installation by leveraging the E4S Spack build cache.

Recent Progress E4S release 1.0 was announced in November 2019 on the external E4S website [249]. The release supports 50 ST products under Linux x86_64. In February 2020, E4S release 1.1 extended support to both NVIDIA and AMD GPUs with the inclusion of CUDA and ROCm in a single image under Linux x86_64. Release 1.1 also introduced support for the Linux ppc64le platform that supports CUDA 10.1. E4S releases contain HPC as well as AI/ML software including TensorFlow and PyTorch. The E4S DocPortal, accessible from the E4S website, was created to rake information from E4S product GitHub pages and provide it in a single location with the most up-to-date information about releases, installation instructions, etc. The E4S validation testsuite [250] was introduced with support for LLVM and other ST products.

In November 2021, E4S v21.11 included x86_64 and ppc64le Docker and Singularity images with 91 E4S products. It included the DOE fork of LLVM compilers as well as E4S products built using these compilers. E4S images with support for three GPU architectures (Intel, AMD, and NVIDIA) are now available for download on DockerHub under the ecpe4s area and are released on the E4S website. These images support Intel oneAPI, AMD ROCm, and NVIDIA NVHPC and CUDA. The E4S Spack Build Cache now includes binaries for ppc64le as well as x86_64 and includes over 77,000 total binaries (Figure 9). E4S containers now support custom images for ECP applications such as WDMapp (Figure 88), Nalu-Wind (Figure 89), and the Pantheon project (Figure 90). The WBS 2.3.6.01 LANL ATDM Software Technologies project highlights the integration of Nalu-Wind and Cinema in a curated Pantheon workflow using an E4S build cache. The E4S build cache has improved the build times for these codes significantly.

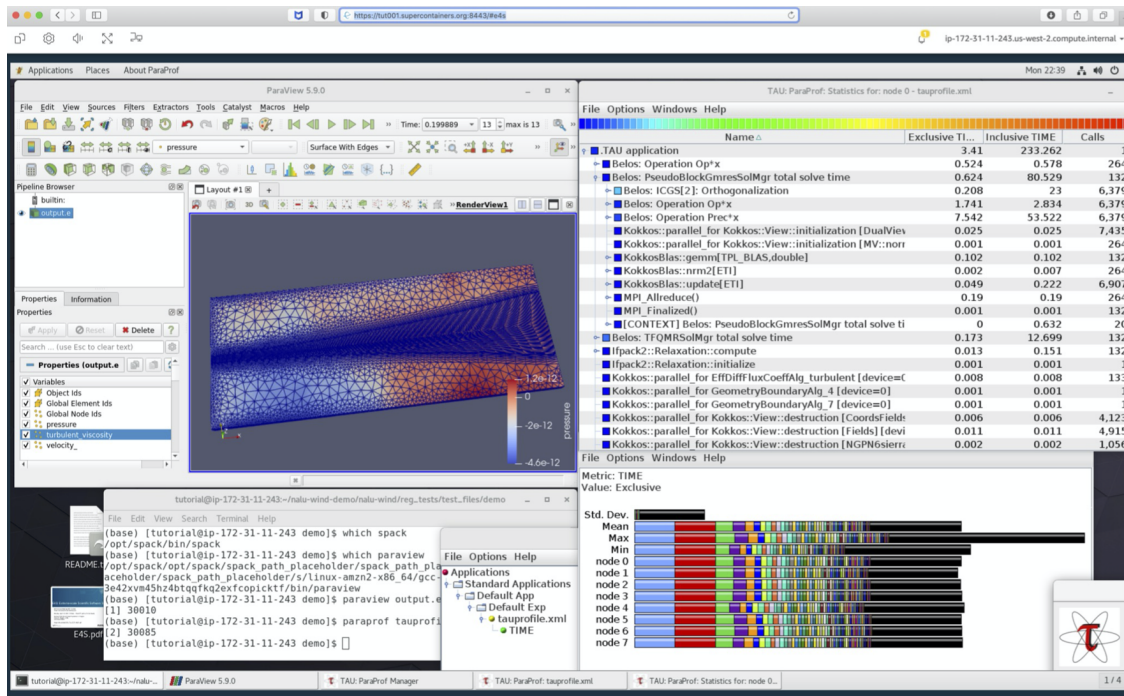


Figure 89: Output from the Nalu-Wind application built using E4S, is visualized using ParaVer and TAU on an AWS cloud instance.

E4S is also deployed on AWS and is used for outreach activities including tutorials at conferences such as ISC-HPC and SC. Figure 89 shows the output of the ECP ExaWind application, Nalu-Wind, being visualized using ParaView. Performance data for the run is being visualized using TAU’s ParaProf browser with instrumentation hooks inserted in the Kokkos runtime. Spack, Trilinos, Kokkos, TAU, ParaVer, Cinema, and Nalu-Wind are all pre-installed in the E4S AWS image.

Also in October 2020, Version 1 of the E4S Community Policies was announced [13]. The E4S Community Policies serve as membership criteria for E4S member packages. The E4S Community Policies have their genesis in the xSDK Community Policies and have a similar purpose—to help address sustainability and interoperability challenges within the complex software ecosystem of which ECP ST is a part.

The process of establishing Version 1 of the E4S Community Policies was a multi-year effort led by the ECP SDK team, including representation from Programming Models and Runtimes, Development Tools, Math Libraries, Data and Vis, and Software Ecosystem and Delivery. This team reviewed the existing xSDK Community Policies and selected those policies that were most generally applicable to all of ECP ST, and not specific to math libraries. From there, the chosen policies were refined and gaps were analyzed.

In addition to the policies shown in Figure 11 a second list of Future Revision policies was also created. These policies are not currently E4S membership criteria, but will be very seriously considered in future versions. In most cases, these policies require further refinement or planning prior to adoption. The topics that these policies address provide information about likely subject areas for E4S policies going forward and are critical to ongoing communication with the E4S community.

In October 2021, Version 2 of the mapping of products to SDKs was completed. This activity was also led by the SDK team, with significant input from ST leadership. The new list includes several changes to existing SDKs, as well as a new Scientific Workflows SDK. The full listing can be seen in Figure 12.

In December 2021, a light-weight measurement of Software Technologies product compatibility with the E4S Community Policies took place during the Software Technologies annual review. The results will help guide efforts to increase the number of products that are compatible with the community policies.

Next Steps Current and near-term efforts include assisting ST leadership in documenting and verifying compatibility with E4S Community Policies; assisting with E4S deployment to computing facilities; adding

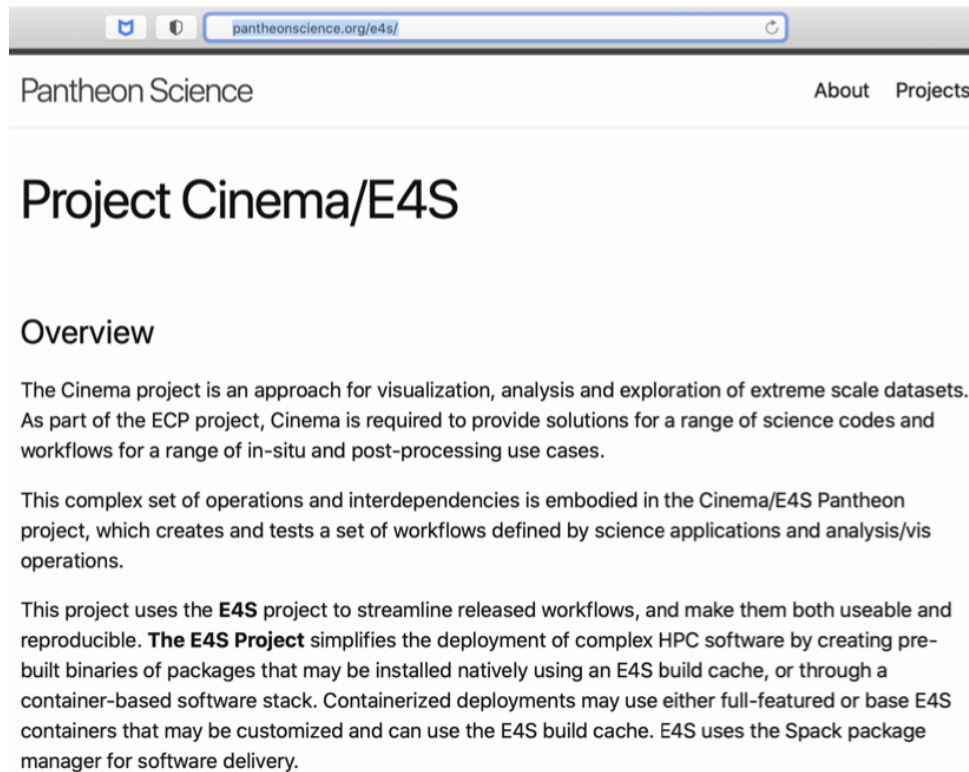


Figure 90: Pantheon Science uses E4S build caches to speed up installation on Summit at ORNL.

additional ST software to E4S; establishing workflows around the maintenance of Spack environments and multipackage CI builds at computing facilities; beginning the work on Version 2 of the E4S Community Policies; and supporting SDK-specific efforts focused on the needs of each SDK with a particular emphasis on sustainability. Preliminary data has been gathered, and a deeper assessment is planned for product teams that want to demonstrate compatibility with the full set of community policies.

4.5.8 WBS 2.3.5.09 Software Packaging Technologies

Overview ECP is tasked with building the first capable exascale ecosystem, and the foundation of this ecosystem, per ECP's mission statement, is an integrated software stack. Building and integrating software for supercomputers is notoriously difficult, and an integration effort for HPC software at this scale is unprecedented. Moreover, the software deployment landscape is changing as containers and supercomputing-capable software package managers like Spack emerge. Spack holds the promise to automate the builds of all ECP software, and to allow it to be distributed in new ways, including as binary packages. Containers will enable entire application deployments to be packaged into reproducible images, and they hold the potential to accelerate development and continuous integration (CI) workflows.

This project will build the tooling required to ensure that packaging technologies can meet the demands of the ECP ecosystem. The project provides Spack packaging assistance for ST users and ECP facilities, and it develops new capabilities for Spack that enable automated deployments of software at ECP facilities, in containerized environments, and as part of continuous integration. Concurrently, the Supercontainers sub-project is investigating and developing technologies and best practices that enable containers to be used effectively at ECP facilities. Supercontainers will ensure that HPC container runtimes will be scalable, interoperable, and integrated into Exascale supercomputing across DOE.

Key Challenges Historically, building software to run as fast as possible on HPC machines has been a manual process. Users download source code for packages they wish to install, and they build and tune it manually for high performance machines. Spack has automated much of this process, but it still requires that users build software. Spack needs modifications to enable it to understand complex microarchitecture details, ABI constraints, and runtime details of exascale machines. This project will enable binary packaging, and it will develop new technologies that enable the same binary packages to be used within containers or in bare metal deployments on exascale hardware.

The Supercontainer effort faces similar challenges to deploying containers on HPC machines. Container technology most notably enables users to define their own software environments, using all the facilities of the containerized host OS. Users can essentially bring their own software stack to HPC systems, and they can snapshot an entire application deployment, including dependencies, within a container. Containers also offer the potential for portability between users and machines. The goal of moving an HPC application container from a laptop to a supercomputer with little or no modification is in reach, but there are a number of challenges to overcome before this is possible on Exascale machines. Solutions from industry, such as Docker, assume that containers can be built and run with elevated privileges, that containers are isolated from the host network, file system, and GPU hardware, and that binaries within a container are unoptimized and can run on any chip generation from a particular architecture. These go against the multi-user, multi-tenant user environment of most HPC centers, and optimized containers may not be portable across systems.

Solution Strategy The Spack project supports ST teams by developing portable build recipes and additional metadata for the ECP package ecosystem. The end goal is to provide a packaging solution that can deploy on bare metal, in a container, or be rebuilt for a new machine on demand. Spack bridges the portability divide with portable package recipes; specialized packages can be built per-site if needed, or lowest-common denominator packages can be built for those cases that do not need highly optimized performance. Packages are relocatable and can be used outside their original build environment. Moreover, Spack provides environments that enable a number of software packages to be deployed together either on an HPC system or in a container.

The Supercontainer project seeks to document current practice and to leverage existing container runtimes, but also to develop new enabling technologies where necessary to allow containers to run on HPC machines. Several HPC container runtimes (e.g., Shifter, Charliecloud, Singularity) already exist, and this diversity enables wide exploration of the HPC container design space, and the Supercontainers project will work with their developers to address HPC-specific needs, including container and job launch, resource manager integration, distribution of images at scale, use of system hardware (storage systems, network and MPI libraries, GPUs and other accelerators), and usability concerns around interfacing between the host and container OS (e.g., bind-mounting, etc. required for hardware support).

The project will document best practices to help educate new users and developers to the advantages of containers, and to help ensure efficient container utilization on supercomputers. Both of these will be living documents, periodically updated in response to lessons learned and feedback. In addition, we will identify gaps, and implement changes and automation in one of the three existing runtimes, as needed. The project will also interface with the E4S and SDK teams, as well as AD teams interested in containerizing their applications. We will work to enable these teams to deploy reproducible, minimally-sized container images that support multiple AD software ecosystems.

Recent Progress Recent progress in this area includes several key developments. Spack’s concretizer was completely reworked, and new code was developed to handle bootstrapping it from binary packages. Pull request testing was implemented across the entire Spack project. Packages are built on each pull request by using cloud infrastructure and ECP EA architecture runners. This effort also worked with the E4S and ST teams to ensure E4S packages can build on EA systems. A GitHub bot (spackbot) was developed to automatically notify maintainers of changes to their packages and to interact with CI pipelines. Multiple container runtimes were tested on pre-exascale systems. Finally, unprivileged container builds were investigated, and the results were accepted as an SC21 paper, “Minimizing privilege for building HPC containers.”

Next Steps A series of efforts have been identified for the next phase of the project. The team will further integrate testing with CI and rework the test dependency model to support more easily building tests after installation. The team will continue to work with the E4S team to provide support for packaging issues on deployed exascale systems. Spack environments will be improved to increase build reproducibility. The team will work with Cray/HPE to improve Spack support for the Cray programming environment. Container deep dives with application teams will be conducted to demonstrate successful and scalable deployment on EA and exascale machines. Finally, container training sessions, outreach efforts, and tutorials will continue.

4.5.9 WBS 2.3.5.10 ExaWorks

Overview Exascale computers will offer transformative capabilities to combine data-driven and learning-based approaches with traditional simulation applications to accelerate scientific discovery and insight. These software combinations and integrations, however, are difficult to achieve due to challenges coordinating and deploying heterogeneous software components on diverse and massive platforms. The ExaWorks project is addressing these challenges by co-designing an open Software Development Kit (SDK) consisting of workflow management systems (WMSs) that can be composed and interoperate through common interfaces. The project is working to bootstrap the SDK with an initial set of tools, design common interfaces, make tools easier to apply to complex science challenges, and apply the SDK to ECP applications. The project is community centered, working with stakeholders, such as users, developers, and facility representatives, to sustainably address workflow requirements at exascale.

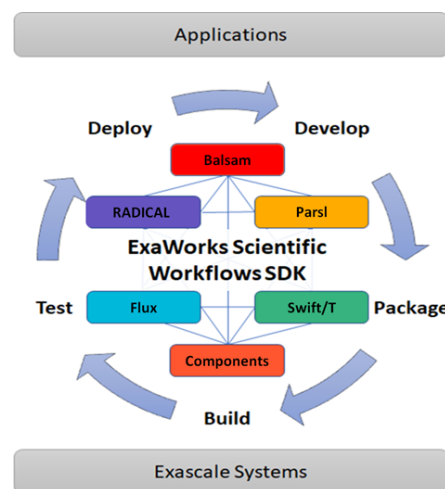


Figure 91: ExaWorks tool kit.

Key Challenges Emerging exascale workflows pose significant challenges to the creation of portable, repeatable, scalable, and performant workflows. These challenges are both technical and non-technical. On the technical side, WMSs are incapable of supporting heterogeneous co-scheduled and high-throughput workflows, and enabling communication between fine-grained tasks in dynamic workflows. On the non-technical side, the myriad WMSs that exist, absence of reusable WMS components, and lack of user guidance when selecting a WMS has led to a disjoint workflows community that tends toward building ad hoc or bespoke solutions rather than adopting and extending existing solutions. Important challenges are outlined below.

Workflows Community The workflows, applications, and facility communities are disjoint. Efforts are needed to bring these groups together to define, develop, and integrate common workflow components.

Scheduling Exascale workflows must manage the efficient execution of diverse tasks (e.g., in runtime, resource requirements, single/multi-node) with complex interdependencies on heterogeneous resources.

Scale and Performance Emerging workflows feature huge ensembles of short-running jobs, which can create millions or even billions of tasks that need to be rapidly scheduled and executed.

Coordination and Communication Workflows depend on coordination between the workflow and the tasks within it, a need that requires efficient exchange of data following various communication patterns.

Portability WMSs are tested on a handful of systems and the frequency by which system hardware and software change makes it impossible to guarantee that a workflow will work on a system in the future.

Solution Strategy The ExaWorks project lays the foundation for an inherently new approach to workflows: establishing an SDK (Figure 91) by assembling components from existing WMSs and defining new component interfaces. The ExaWorks SDK provides a robust, well-tested, well-documented, and scalable set of tools and

components that can be combined to enable diverse teams to produce scalable and portable workflows for a wide range of exascale applications. Importantly, the project is not creating a new workflow system nor does it aim to replace the many workflow solutions already deployed and used by scientists, but rather it will provide a well-engineered and scalable SDK which can be leveraged by new and existing workflows.

The goals of the first year of the project are to instantiate the ExaWorks SDK, seed the SDK with robust workflow component technologies, explore pairwise integrations between components, and define common component interfaces. The project also aims to impact ECP applications and bring together the workflows community, including developers, ECP applications, and DOE compute facility representatives to collectively address workflows challenges.

Recent Progress In this period of the project, we focused on addressing our goals outlined above. We briefly describe efforts in each area.

We released the first version of the ExaWorks SDK. We first defined community policies [251] (modeled on E4S) and developed a GitHub Actions-based continuous integration model and a Spack-based packaging approach. Our packaging solutions allow for portable and reproducible deployment and testing techniques, which vary considerably among WMSs. We have bootstrapped the SDK with five workflow technologies that are widely used in ECP applications. We followed a community-based approach in which all artifacts are tracked using an open process on GitHub.

Based on feedback from the ECP workflows community, we identified our first component interface for the SDK: PSI/J, a portability layer across different HPC workload managers. As a community, we defined a language-agnostic specification [252] and after iteration released a first version of the specification. We have also released a 0.1 version of a Python implementation [253]. The Python library contains the set of core classes defined by the specification, as well as abstract base classes (ABCs) for components that implement functionality specific to different clusters and batch systems. The implementation currently supports Slurm, LSF, Cobalt, RADICAL-Pilot, and Flux connectors.

We have continued to work closely with several ECP applications, including ExaLearn, CANDLE, and ExaAM. As the ExaAM workflow is in its early stages. We are using this engagement as a way of defining processes for engaging with new ECP teams in the future. The ExaAM workflow is composed of several application codes, each capturing a different scale or physics of the problem. Each stage feeds information to the next stage, and iteration can occur across stages. Some stages leverage CPU's and others GPU's. Several stages are themselves small workflows typically leveraging the ensemble motif. Thus, the ExaAM workflow requires integration of multiple physics applications and when executed at full scale will require complex orchestration of many sub-workflows. Initial integration of ExaWorks SDK technologies has shown that improvements in throughput are possible with minor changes to existing scripts.

In collaboration with the NSF-funded WorkflowsRI project, we are hosting a series of workflows community summits that aim to bring the workflows community together [254]. The first summit [255] brought together 48 international participants representing many WMSs, with the goal to identify crucial challenges in the workflows community. The summit considered six broad themes: FAIR workflows, training and education, AI workflows, exascale challenges, APIs and interoperability, and developing a workflow community. The second summit [256] focused on technical approaches for realizing many of the challenges identified in the first summit. It included 75 workflows developers and focused on three core topics: defining common workflow patterns and benchmarks, identifying paths toward interoperability of workflow systems, and improving workflow systems' interface with legacy and emerging HPC software and hardware. The third workshop (scheduled for early November 2021) will explore workflows challenges and opportunities from the perspective of computing centers and facilities. It will bring together a small group of facilities representatives with the aim to understand how workflows are currently being used at each facility, how facilities would like to interact with workflow developers and users, how workflows fit with facility roadmaps, and what opportunities there are for tighter integration between facilities and workflows. We also host an open monthly community meeting that is attended by representatives across DOE laboratories. A snapshot of progress and plans of ExaWorks was accepted for the SC21 workshop on workflows (WORKS21) [257].

Next Steps The next steps for the SDK are to develop and harden deeper integration of our technologies, to make our SDK available via E4S, to extend our CI/CD pipeline to ECP systems, and to integrate other community workflows tools. Thus far, we have focused on small-scale interoperation testing, but we will

extend our solutions to ensure scalable integration of the components on pre-exascale and early access exascale systems for immediate readiness as exascale systems become available. We will complete a first version of the PSI/J Python library and work to integrate it in various WMSs.

For applications engagement, we plan to build on the ExaAM workflow to explore and showcase multiple SDK technologies, and to bootstrap integrated testing of the components in a realistic workflow. We will leverage this engagement to gain real-world experience in how the SDK can be leveraged by application teams and integrated with CI and other infrastructures. We also plan to apply our SDK to ECP applications to address broad workflow needs.

We are preparing an ExaWorks tutorial at SC21. We will incorporate the experiences and lessons learned from this tutorial into our user-facing SDK documents.

4.6 WBS 2.3.6 NNSA ST

End State: In the end, the goal is to have software used by the NNSA/ATDM national security applications and associated exascale facilities be hardened for production use and made available to the larger ECP community.

4.6.1 *Scope and Requirements*

The NNSA ST L3 area was created in FY20, although the projects included have all been part of the ECP before its creation. The capabilities of these software products remains aligned with the other ST L3 areas from which they were derived, but are managed separately for non-technical reasons out of scope of this document.

The resulting products in this L3 area are open source, important or critical to the success of the NNSA National Security Applications, and are used (or potentially used) in the broader ECP community. The products in this L3 span the scope of the rest of ST (Programming Models and Runtimes, Development Tools, Math Libraries, Data Analysis and Vis, and Software Ecosystem), and will be coordinated with those other L3 technical areas through a combination of existing relationships and cross-cutting efforts such as the ST SDKs and E4S.

4.6.2 *Objectives*

The objective of these software products is to support the development of new from-scratch applications within the NNSA that were started just prior to the founding of the ECP under the ATDM program element within NNSA and ASC. While earlier incarnations of these products may have been more research-focused, by the time of the ECP ST restructuring in 2019 that resulted in this L3 area, these products are in regular use by their ATDM applications, and have matured to the point where they are ready for use within the broader open source community.

4.6.3 *Plan*

NNSA ST products are developed with and alongside a broader portfolio of ASC products in an integrated program, and are planned out at high level in the annual ASC Implementation Plan, and in detail using approved processes within the home institution/laboratory. They are scoped to have resources sufficient for the success of the NNSA mission, as well as a modicum of community support (e.g., maintaining on GitHub, or answering occasional questions from the community).

For ECP products not part of the NNSA portfolio that have critical dependencies on these products, there are often other projects within ECP that provide additional funding and scope for those activities. In those cases, there may be additional information within this document on these products.

4.6.4 *Risks and Mitigation Strategies*

One risk associated with the NNSA ST projects is the programming environment of the El Capitan system. The programming environment on this system will be a departure from what the NNSA software teams have used before, so there is a risk that it will present challenges that cause delays in porting the software to the

system. That said, the probability of this risk is low because the programming environment of El Capitan will also be installed on DOE predecessor machines so it is likely that it challenges will be identified and addressed by the time of El Capitan. NNSA ST projects can mitigate this risk by evaluating their software on the predecessor systems to identify challenges early.

Another risk associated with the NNSA ST L3 is that the projects rely on multiple sources of funding outside of ECP. The budget priorities of those external sources may not always be aligned with those of ECP. In general, this risk is low because the L4 leads strive to align their project goals across all funding sources. However, it is possible that funding expected to be leveraged to develop a feature to later be used for ECP purposes may be dropped. If this occurs, the L4 leads will need to mitigate the situation according to their individual project needs, perhaps by renegotiating deliverable time lines.

Another risk is that others in the community will pick up these products as open source, and expect additional support beyond the scope of the primary NNSA mission. If those dependent products are within the ECP, the main mitigation is to use ASCR contingency funding to provide additional development and support - potentially through support of teams outside of the home institution. If those dependent products are in the broader community, then mitigations are generally outside of the scope of the ECP, although each NNSA lab typically has some sort of project (or possibly even a policy) on how to deal with external demands on open source products.

4.6.5 WBS 2.3.6.01 LANL ATDM Software Technologies

Overview The LANL ATDM PMR effort is focusing on the development and use of advanced programming models for (ATDM) use-cases. Our current focus is on research and development of new programming model capabilities in the Legion data-centric programming system. Legion provides unique capabilities that align well with our focus on the development of tools and technologies that enables a separation of concerns of computational physicists and computer scientists. Within the ATDM PMR effort we have focused on the development of significant new capabilities within the Legion runtime that are specifically required to support LANL's ATDM applications. Another key component of our work is the co-design and integration of advanced programming model research and development within FleCSI, a Flexible Computational Science Infrastructure. A major benefit to the broader ECP community is the development of new features in the Legion programming system which are available as free open-source software.²⁴

The Kitsune project, provides a compiler-focused infrastructure for improving various aspects of the exascale programming environment. At present, efforts are primarily focused on advanced LLVM compiler and tool infrastructure to support the use of a *parallel-aware* IR. In addition, we are actively involved in the Flang Fortran front-end that is now an official sub-project within the overall LLVM infrastructure. All these efforts include interactions across ECP as well as with the broader LLVM community and industry.

The LANL ATDM Cinema project develops scalable solutions for data analysis as part of the Data and Visualization software stack. Cinema is a novel database approach to saving data extracts in situ which are then available for post hoc interactive exploration. These data extracts can include metadata, parameters, data visualizations, small meshes, output plots, etc. Cinema workflows enable flexible data analysis using a fraction of the file storage. Cinema ECP workflows that integrate applications, in situ and post-processing analysis are captured and curated through the Pantheon project, which is focused on reproducible ECP workflows. By integrating E4S Spack build caches [258] of both applications and dependent capabilities (Ascent, etc.), Pantheon workflows can be downloaded, built and run quickly enough to be useful in a variety of applications such as CI, prototyping functionality or testing analyses.

The BEE/Charliecloud subproject is creating software tools to increase portability and reproducibility of scientific applications on high performance and cloud computing platforms. Charliecloud [259] is an unprivileged Linux container runtime. It allows developers to use the industry-standard Docker [260] toolchain to containerize scientific applications and then execute them on unmodified DOE facility computing resources without paying any performance penalty. BEE [261] (Build and Execution Environment) is a toolkit providing users with the ability to execute application workflows across a diverse set of hardware and runtime environments. Using Bee's tools, users can build and launch applications on HPC clusters and public and private clouds, in containers or in containers inside of virtual machines, using a variety of container runtimes such as Charliecloud and Docker.

²⁴<https://gitlab.com/StanfordLegion/legion>

Key Challenges Each of the applications and programming models described above faces key challenges to their successful implementation and deployment.

Legion Applications will face significant challenges in realizing sustained performance on next-generation systems. Increasing system complexity coupled with increasing scale will require significant changes to our current programming model approaches. This is of particular importance for large-scale multi-physics applications where the application itself is often highly dynamic and can exhibit high variability in resource utilization and system bottlenecks depending on what physics are currently in use (or emphasized). Our goal in the LANL ATDM PMR project is to support these highly dynamic applications on Exascale systems, providing improvements in productivity, long-term maintainability, and performance portability of our next-generation applications.

FleCSI Legion Integration FleCSI is a Flexible Computational Science Infrastructure whose goal is to provide a common framework for application development for LANL’s next-generation codes. FleCSI is required to support a variety of different distributed data structures and computation on these data structures including structured and unstructured mesh as well as mesh-free methods. Our work in the LANL ATDM PMR project is focused on co-designing the FleCSI data and execution model with the Legion programming model to ensure the latest advancements in the programming model and runtimes research community are represented in our computational infrastructure. A significant challenge in our work is the additional constraint that FleCSI must also support other runtime systems such as MPI. Given this constraint, we have chosen an approach that ensures functional correctness across both runtimes but that also leverages and benefits from capabilities in Legion that are not directly supported in MPI (such as task-based parallelism as a first-class construct).

Kitsune A key challenge to our efforts is reaching agreement within the broader community that a parallel IR is beneficial and needed within LLVM. This not only requires showing the benefits but also providing a full implementation for evaluation and feedback from the community. In addition, significant new compiler capabilities represent a considerable effort to implement and involve many complexities and technical challenges. These efforts and the process of up-streaming potential design and implementation changes do involve some amount of time and associated risk.

Additional challenges come from a range of complex issues surrounding target architectures for exascale systems. Our use of the LLVM infrastructure helps reduce many challenges here since many processor vendors and system providers now leverage and use LLVM for their commercial compilers.

Cinema Interfacing to a large number of ECP applications with the Cinema software and the management of the voluminous data from these applications.

Bee/CharlieCloud Other HPC-focused container runtimes exist, such as NERSC’s Shifter [262] and Singularity [263]. These alternative runtimes have characteristics, such as complex setup requirements and privileged user actions, that are undesirable in many environments. Nevertheless, they represent a sizable fraction of the existing HPC container runtime mindshare. A key challenge for BEE is maintaining support for multiple runtimes and the various options that they require for execution. This is especially true in the case of Singularity, which evolves rapidly. Similarly, there is a diverse collection of resources that BEE and Charliecloud must support to serve the ECP audience. From multiple HPC hardware architectures and HPC accelerators such as GPUs and FPGAs, to differing HPC runtime environments and resource managers, to a multitude of public and private cloud providers, there is a large set of available resources that BEE and Charliecloud must take into consideration to provide a comprehensive solution.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each effort.

Legion In funded collaboration with NVIDIA, LANL and NVIDIA are developing new features in Legion to support our applications. Necessary features are identified through direct engagement with application developers and through rapid development, evaluation, and refactoring within the team. Major features include Dynamic Control Replication for improved scalability and productivity and Dynamic Tracing to reduce runtime overheads for applications with semi-regular data dependencies such as applications with stencil-based communication patterns (Figure 92).

FleCSI Legion Integration LANL staff work on co-design and integration of the Legion programming system into the FleCSI framework. We have regular milestones that align well with application needs and the development of new features within Legion.

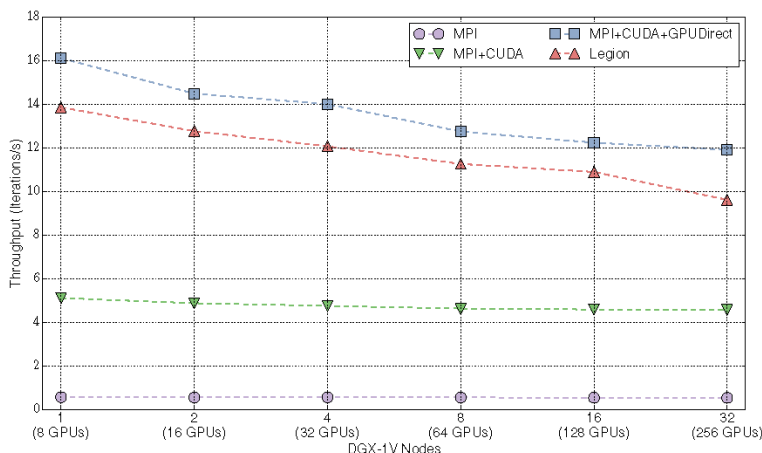


Figure 92: Productivity features such as dynamic control replication scales well across multiGPU systems in unstructured mesh computations.

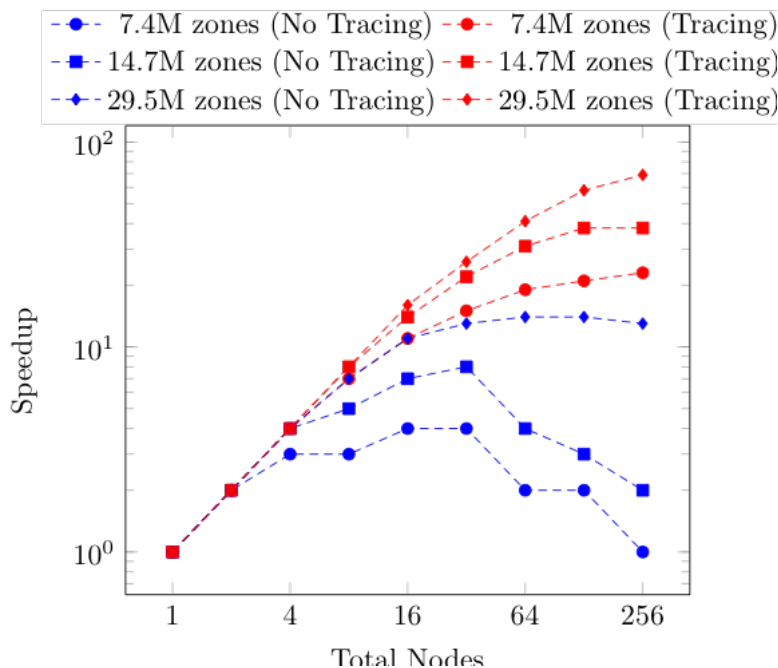


Figure 93: New Legion features such as tracing will improve strong scaling in unstructured mesh computations.

Kitsune Given the project challenges, our approach takes aspects of today’s node-level programming systems (e.g. Kokkos) and programming languages (e.g. C++) into consideration and aims to improve and expand upon their capabilities to address the needs of ECP and LANL’s mission critical applications. This allows us to attempt to strike a balance between incremental improvements to existing infrastructure and more aggressive techniques that seek to provide innovative solutions, thereby managing risk while also providing the ability to introduce new technologies via the toolchain.

Unlike current designs, our approach introduces the notion of explicit parallel constructs into the LLVM IR, building off of work done at MIT on Tapir [264] and the OpenCILK effort (<https://cilk.mit.edu>). We are working with MIT to extend this work as well as making some changes to fundamental data structures within the LLVM infrastructure to assist with and improve analysis and optimization passes.

Cinema Cinema [265] is being developed in coordination with LANL’s ECP application NGC to ensure that data collected during the simulation execution is of appropriate frequency, resolution, and view port for later analysis and visualization by scientists. Cinema is essential for ECP because it embodies approaches to maximize insight from extreme-scale simulation results while minimizing data footprint. Cinema capabilities provide scientists more options in analyzing and exploring the results of large simulations by providing a workflow that (1) detects features in situ, (2) captures data artifacts in Cinema databases, (3) promotes post-hoc analysis of the data, and (4) provides data viewers that allow interactive, structured exploration of the resulting artifacts. Cinema database export is part of the extract generator workflow in ParaView/Catalyst and Cinema export is also available via ALPINE’s Ascent infrastructure and through VisIt.

Bee/CharlieCloud The BEE/Charliecloud project is focusing first on providing support for containerized production LANL scientific applications across all of the existing LANL production HPC systems. The BEE/Charliecloud components required for production use at LANL will be documented, released and fully supported. Follow-on development will focus on expanding support to additional DOE platforms. This will mean supporting multiple hardware architectures, operating systems, resource managers, and storage subsystems. Support for alternative container runtimes, such as Docker, Shifter, and Singularity is planned.

Recent Progress Recent progress includes developments in Legion, FleCSI Legion integration, Kitsune, Cinema, and Bee/CharlieCloud.

Legion One of the strengths of Legion is that it executes asynchronous tasks as if they were executed in the sequence they occur in the program. This provides the programmer with a mental model of the computation that is easy to reason about. However, the top-level task in this tree-of-tasks model can often become a sequential bottleneck, as it is responsible for the initial distribution of many subtasks across large machines. In earlier work NVIDIA developed the initial implementation of control replication, which allows the programmer to write tasks with sequential semantics that can be transparently replicated many times, as directed by the Legion mapper interface, and run in a scalable manner across many nodes. Dynamic control replication is an important capability for LANL’s ATDM effort, allowing our application teams to write applications with apparently sequential semantics while enabling scalability to Exascale architectures. This approach will improve understandability of application code, productivity, and composability of software and ease the burden of optimization and porting to new architectures. New dynamic tracing ability has been added to Legion to allow debugging and insight in to performance optimization activities (Figure 93).

FleCSI Legion Integration A key component of LANL’s Advanced Technology Development and Mitigation effort is the development of a flexible computational science infrastructure (FleCSI) to support a breadth of application use cases for our Next Generation Code. FleCSI has been co-designed with the Legion programming system in order to enable our Next Generation Code to be performance portable and scalable to future Exascale systems. Legion provides the underlying distributed and node-level runtime environment required for FleCSI to leverage task and data parallelism, data dependent execution, and runtime analysis of task dependencies to expose parallelism. We completed testing of Legion on Sierra with a Visco-Plastic Self-Consistent, VSCP, application to investigate initial performance on GPU systems.

Kitsune Our primary focus is the delivery of capabilities for LANL’s ATDM Ristra application (AD 2.2.5.01). In support of the requirements for Ristra, we are targeting the lowering of `forall` constructs, including Kokkos `parallel_for` construct, directly into the parallel representation. At present, this works for many C++ constructs (e.g., `for` and `for-range` statements). We can target this code to different runtimes and architectures via the compiler and thus avoid reimplementing of Kokkos or fundamental C++ constructs. In addition we are working to replace LLVM’s dominator tree, a key data structure for optimizations including parallelization and memory usage analysis, with a dominator directed-acyclic-graph (DAG). This capability is still in its early evaluation state and we continue to explore correctness and compatibility within the

overall LLVM infrastructure. We are actively watching recent events within the LLVM community around multi-level intermediate representations (MLIR) and the relationship they have with parallel semantics, analysis, optimization, and code generation.

At present we are successfully compiling our full applications using the new toolchain. We continue to test, debug, and work towards improved optimizations and performance.

Cinema Recent Cinema progress this past year has focused on expanding python-based Cinema functionality and the development of the Composable Image Set (CIS) specification. This new CIS specification can be used to save layers of float-value png images that can be combined to generate composited visualizations post hoc in a more flexible and interactive approach. An example of the new CIS format is shown in Figure 94 using a groundwater data set [266]. The two materials in the simulation—stone and water streamlines—are separately saved. Post hoc, a visualization can be generated by colormapping and shading each layer. Different layers can be combined to form a composited visualization.

The Cinema Python module, *cinemasci*, has been successfully deployed and passes unit testing on Spock.

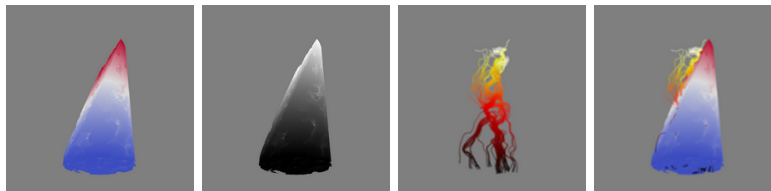


Figure 94: In this example of the new Cinema CIS format, a groundwater simulation can be split into water streamlines and stone layers. From left to right: the stone layer has a cool-warm colormap applied post hoc; the same stone layer in grayscale; the water streamlines are colormapped in inferno; the stone layer and the water streamline layers are separately colormapped and then composited to form the final visualization image.

The Cinema team has also been collaborating with the E4S team to leverage the E4S SDKs and cached builds to generate reproducible ECP workflows curated via the Cinema/E4S Pantheon project [267]. These reproducible workflows provide end-to-end pipelines that can demonstrate integrations. Leveraging Spack-based builds, Pantheon curates the set of tags needed for a working pipeline on a specific exascale platform. Workflows are saved as a git repository and can be reliably cloned and run. Figure 95 shows the steps in a Nalu-Wind [268] Pantheon workflow [269] that results in a validated Cinema database.

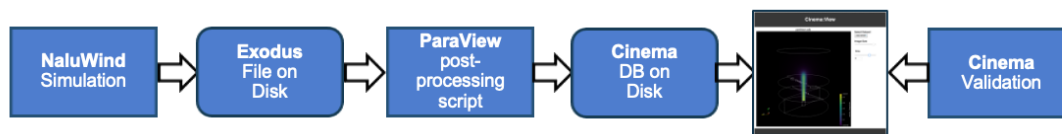


Figure 95: In this example of a curated Pantheon workflow, the Nalu-Wind simulation outputs a data file which is input to a ParaView script that outputs a Cinema database of visualizations that then go through a validation step to verify the correctness of the Cinema database.

Bee/CharlieCloud Recent Charliecloud progress has focused on understanding and documenting best practices for running large scale MPI jobs using containerized runtimes. Charliecloud is enhancing support for multiple MPI implementations. Charliecloud is available at <https://github.com/hpc/charliecloud> and is distributed inside of Debian and Gentoo Linux distributions as well as being part of OpenHPC. Charliecloud won an 2018 R&D 100 award.

BEE fully supports launching Charliecloud containers on all LANL HPC systems. It can also launch containers on AWS and OpenStack clouds such as NSF Chameleon. BEE also supports interactive launching of jobs with the SLURM resource manager. BEE was shown at the end of FY19 to support a complex

multiphysics application with setup, in situ visualization and checkpoint-restart on a production system at LANL.

Next Steps The next steps for each effort have already been identified.

Legion Focus on hardening and scalability of Legion’s Dynamic Control Replication and development of Dynamic Tracing for application use-cases.

FleCSI Legion Integration Support the Ristra Application milestone to run on Sierra and Trinity.

Kitsune The key next steps for our efforts are to expand our test cases by increasing the complexity of the codes we’re compiling, supporting additional forms of Kokkos constructs, and support other parallel constructs that meet the needs of Ristra. Where possible, we will explore a broader set of use cases within the ECP community. This will be done while also striving to maintain a feature set in Kitsune that matches the most recent releases of the LLVM infrastructure. We will continue a quarterly release cycle of the software and also when feature sets align with our milestones. This work will go hand-in-hand with the code generation and optimization for the exascale system target processors: including CPUs and GPUs on the target platforms. The development of associated runtime targets that can reduce code generation complexity will also be a component of our future efforts (as needed).

With the addition to Fortran to LLVM via the Flang front end we will also look to add support for lowering to the parallel IR in those use cases.

Cinema In FY22, Cinema will be working with our ST partners to deploy additional Cinema functionality, exploring ECP use cases for the new CIS format and delivering reproducible Pantheon workflows that encapsulate integrations with Cinema partners.

Bee/CharlieCloud A refactoring of BEE to support an open standard is underway. Support for the Open Workflow standard will allow a base on a well defined workflow description language leveraged by other scientific communities. This will then be tested on multiple systems to ensure portability.

4.6.6 WBS 2.3.6.02 LLNL ATDM Software Technologies

Overview Spack is a package manager for HPC [270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284]. It automates the process of downloading, building, and installing different versions of HPC applications, libraries, and their dependencies. Facilities can manage multi-user software deployments, and developers and users can manage their own stacks separately. Spack enables complex applications to be assembled from components, lowers barriers to reuse, and allows builds to be reproduced easily.

The MFEM library [285, 286] is focused on providing high-performance mathematical algorithms and finite element discretizations to next-gen high-order ECP/ATDM applications. A main component of these efforts is the development of ATDM-specific physics enhancements in the finite element algorithms in MFEM and the MFEM-based BLAST Arbitrary Lagrangian-Eulerian (ALE) code [287], in order to provide efficient discretization components for LLNL’s ATDM efforts, including the MARBL application (ECP’s LLNLApp).

A second main task in the MFEM project is the development of unique unstructured adaptive mesh refinement (AMR) algorithms in MFEM, that focus on generality, parallel scalability, and ease of integration in unstructured mesh applications. The new AMR capabilities can benefit a variety of ECP apps that use unstructured meshes, as well as many other applications in industry and the SciDAC program.

Another aspect of the work is the preparation of the MFEM finite element library and related codes for exascale platforms by using mathematical algorithms and software implementations that exploit increasing on-node concurrency targeting multiple complex architectures (e.g. GPUs). This part of the project is synergistic with and leverages efforts from the ECP CEED co-design center.

MFEM is an open-source finite element library with 10000 downloads/year from 100+ countries. It is freely available at <https://mfem.org> and on GitHub <https://github.com/mfem>, where the MFEM community includes more than 450 members, as well as via Spack and OpenHPC. The application outreach and the integration in the ECP ecosystem is further facilitated by MFEM’s participation in ECP’s xSDK and E4S projects.

RAJA, CHAI, and Umpire are providing software libraries that enable application and library developers to meet advanced architecture portability challenges. The project goals are to enable writing performance portable computational kernels and coordinate complex heterogeneous memory resources among components in a large integrated application. These libraries enhance developer productivity by insulating them from much of the complexity associated with parallel programming model usage and system-specific memory concerns.

This project provides three complementary and interoperable libraries:

1. RAJA: Software abstractions that enable C++ developers to write performance portable (i.e., single-source) numerical kernels (loops).
2. CHAI: C++ managed array abstractions that enable transparent and automatic copying of application data to memory spaces at run time as needed based on RAJA execution contexts.
3. Umpire: A portable memory resource management library that provides a unified high-level API in C++, C and FORTRAN for resource discovery, memory provisioning, allocation, transformation, and introspection.

Capabilities delivered by these software efforts are needed to manage the diversity and uncertainty associated with current and future HPC architecture design and software support. Moving forward, ECP applications and libraries need to achieve performance portability: without becoming bound to particular (potentially limiting) hardware or software technologies, by insulating numerical algorithms from platform-specific data and execution concerns, and without major disruption as new machine, programming models, and vendor software become available.

These libraries in development in this project are currently used in production ASC applications at LLNL and receive most of their support from the LLNL national security application project. They are also being used or being explored/adopted by several ECP application and library projects, including: LLNL ATDM application, GEOS (Subsurface), SW4 (EQSIM), MFEM (CEED co-design), DevilRay (Alpine), and SUNDIALS.

Flux [288, 289] is a next-generation workload management framework for HPC. Flux maximizes scientific throughput by scheduling the scientific work requested by HPC users—also known as jobs or workloads. Using its highly scalable breakthrough approaches of fully hierarchical scheduling and graph-based resource modeling, Flux manages a massive number of processors, memory, GPUs, and other computing system resources—a key requirement for exascale computing and beyond. A job is typically expressed in a script that contains a formal specification for resource requests, identifies applications (for instance, multi-physics simulation software to run simultaneously across resources) along with their input data and environment, and describes how to deliver the output data. Modern scientific computing campaigns contain numerous interconnected and dependent tasks with disparate resource requirements. The composition of these numerous interdependent tasks can be spread across many jobs, as well as within each job, and is often referred to as a scientific workflow (as distinct from a single job or workload). Scheduling resources to such modern workflows requires highly scalable scheduling as well as high-performance communication and coordination across hundreds of thousands of jobs, which could not be accomplished with traditional HPC schedulers. Flux solves this critical problem and has provided innovative solutions for modern workflows for many scientific and engineering disciplines.

AID (Advanced Infrastructure for Debugging) provides an advanced debugging, code-correctness and testing tool set to facilitate reproducing, diagnosing and fixing bugs within HPC applications. The current capabilities include the following:

- STAT (highly scalable lightweight debugging tool);
- Archer (low-overhead OpenMP data race detector);
- ReMPI/NINJA (scalable record-and-replay and smart noise injector for MPI); and
- FLiT/FPUChecker (floating-point correctness checking tool suite).

Major efforts include developing and deploying additional capabilities within the team’s tool set for exascale systems and integrating them to ASC and ECP/ATDM codes. The team strives to do this through co-design efforts with both large HPC code teams and exascale computing hardware vendors themselves.

Caliper is a program instrumentation and performance measurement framework. It is designed as a performance analysis toolbox in a library, allowing one to bake performance analysis capabilities directly into applications and activate them at runtime. Caliper can be used for lightweight always-on profiling or advanced performance engineering use cases, such as tracing, monitoring, and auto-tuning. It is primarily aimed at HPC applications, but works for any C/C++/Fortran program on Unix/Linux.

Key Challenges Each of the applications and tools described above faces key challenges to their successful implementation and deployment.

Spack Spack makes HPC software complexity manageable. Obtaining optimal performance on supercomputers is a difficult task; the space of possible ways to build software is combinatorial in size, and software reuse is hindered by the complexity of integrating a large number of packages and by issues such as binary compatibility. Spack makes it easy to build optimized, reproducible, and reusable HPC software.

MFEM The key challenges addressed by the LLNL ATDM Mathematical Libraries project are as follows:

Robust high-order finite element methods for ALE compressible flow. Although high-order methods offer significant advantages in terms of HPC performance, their application to complicated ALE problems requires careful consideration to control oscillations and ensure accuracy.

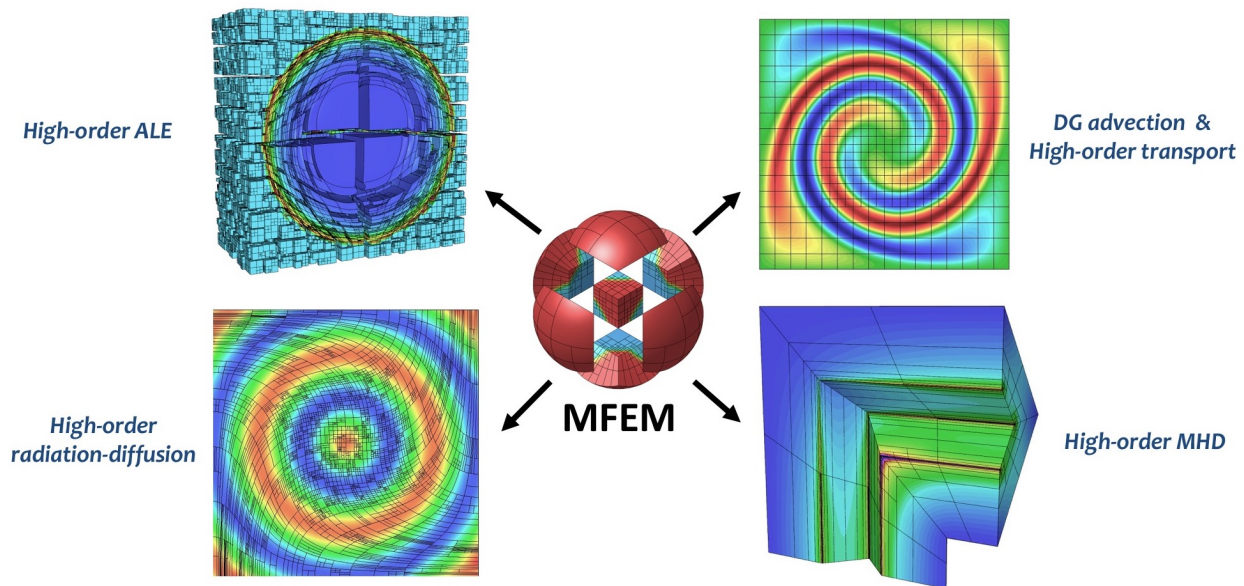


Figure 96: AMR implementation in MFEM allows many applications to benefit from non-conforming adaptivity, without significant changes in their codes.

Scalable algorithms for unstructured adaptive mesh refinement. Adaptive mesh refinement is a common way to increasing application efficiency in problems with localized features. While block-structured AMR has been well-studied, applying AMR in unstructured settings is challenging, especially in terms of derefinement, anisotropic refinement, parallel rebalance and scalability.

GPU porting of finite element codes. Owing to the relatively high complexity of the finite element machinery, MFEM, BLAST and related codes use object-oriented C++ design that allows generality and flexibility, but poses challenges in terms of porting to GPU architectures. Finding the right balance between generality and performance in the GPU context is an important challenge for many finite element-based codes that remains outstanding in the current software and programming model environment.

RAJA, Umpire, and CHAI Exascale machines are expected to be very diverse, with different GPU, threading, memory models, and node architectures. A parallelization strategy that works well for one machine may not work well for another, but application developers cannot afford to develop multiple versions of their code for each machine they support. Rather, the application must be written using higher-level abstractions, and adapted at a lower level, with minimal programmer effort, to specific machines. RAJA, Umpire, and CHAI address this by giving applications the flexibility to adapt and tune for many target machines, using the same high level kernel formulations. In other words they separate the concerns of performance and correctness and avoid a combinatorial explosion of code versions for the exascale ecosystem.

In addition to performance portability, RAJA, Umpire, and CHAI specifically target the porting issues faced by legacy codes. Where other performance portability frameworks may require a larger up-front investment in data structures and code restructuring, RAJA, Umpire, and CHAI are non-invasive and allow codes to adopt strategies for loop parallelism, data layout tuning, and memory management separately. Legacy applications need not adopt all three at once; they can gradually integrate each framework, at their own pace, with a minimal set of code modifications.

Flux The workload manager is responsible for efficiently delivering compute cycles of HPC systems to multiple users while considering their diverse resource types—e.g., compute racks and nodes, central and graphics processing units (CPUs and GPUs), multi-tiered disk storage. However, two technical trends are making even the best-in-class products significantly ineffective on exascale computing systems. The first trend is the evolution of workloads for HPC. With the convergence of conventional HPC with new simulation, data analysis, machine learning (ML), and artificial intelligence (AI) approaches, researchers are ushering in new scientific discoveries. But this evolution also produces computing workflows—often comprising many distinct tasks interacting with one another—that are far more complex than traditional products can sufficiently manage. Second, hardware vendors have steadily introduced new resource types and constraints into HPC systems. Multi-tiered disk storage, CPUs and GPUs, power efficiency advancements, and other hardware components have gained traction in an era in which no single configuration reigns. Many HPC architectures push the frontiers of compute power with hybrid (or heterogeneous) combinations of processors. The workload management software must manage and consider *extremely heterogeneous* computing resources and their relationships for scheduling in order to realize a system’s full potential.

AID Debugging parallel applications running on supercomputers is extremely challenging. Supercomputers may contain very high numbers of compute cores and multiple GPUs, and applications running on such systems must rely on multiple communication and synchronization mechanisms as well as compiler optimization options to effectively utilize the hardware resources. These complexities often produce errors that occur only occasionally, even when run with the exact same input on the same hardware. These so-called non-deterministic bugs are remarkably challenging to catch due in large part to difficulty in reproducing them. Some errors may not even reproduce when being debugged, as the act of debugging may perturb the execution enough to mask the bug. To find and fix these errors, programmers currently must devote a large amount of effort and machine time.

Caliper Caliper addresses the challenges of providing *meaningful* measurements for large applications. Often, measurements of FLOPs, timings, data movement, and other quantities are not associated with key application constructs that give them meaning. For example, we may know the number of floating point instructions over an entire application run, but if we do not know the number of mesh elements or the particular physics phase associated with the measurement, we may be unable to determine whether the FLOPS achieved are good or bad. Caliper separates these concerns: application developers can instrument the phases other context in their code, and performance analysts and users may turn on performance measurements that are then associated with the context. Caliper associates meaning with HPC performance measurements.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each effort.

Spack Spack provides a domain-specific language for templated build recipes. It provides a unique infrastructure called the *concretizer*, which solves the complex constraint problems that arise in HPC dependency resolution. Developers can specify builds *abstractly*, and Spack automates the tedious configuration process

and drives the build. Spack also includes online services to host recipes, code, and binaries for broad reuse. These repositories are maintained by Spack’s very active community of contributors.

MFEM The MFEM team has performed and documented a lot of research in high-performance mathematical algorithms and finite element discretizations of interest to ATDM applications [290, 291, 292, 293, 294, 295, 296, 297]. Our work has demonstrated that the high-order finite element approach can successfully handle coupled multi-material ALE, radiation-diffusion and MHD. We have also shown how high-order methods can be adapted for monotonicity (positivity preservation), handling of artificial viscosity (shock capturing), sub-zonal physics via closure models, etc.

To enable many applications to take advantage of unstructured mesh adaptivity, the MFEM team is developing AMR algorithms at library level, targeting both *conforming* local refinement on simplex meshes and *non-conforming* refinement for quad/hex meshes. Our approach is fairly general, allowing for any high-order finite element space, H1, H(curl), H(div), on any high-order curved mesh in 2D and 3D, arbitrary order hanging nodes, anisotropic refinement, derefinement and parallel load balancing. An important feature of our library approach is that it is independent of the physics, and thus easy to incorporate in apps, see Figure 96.

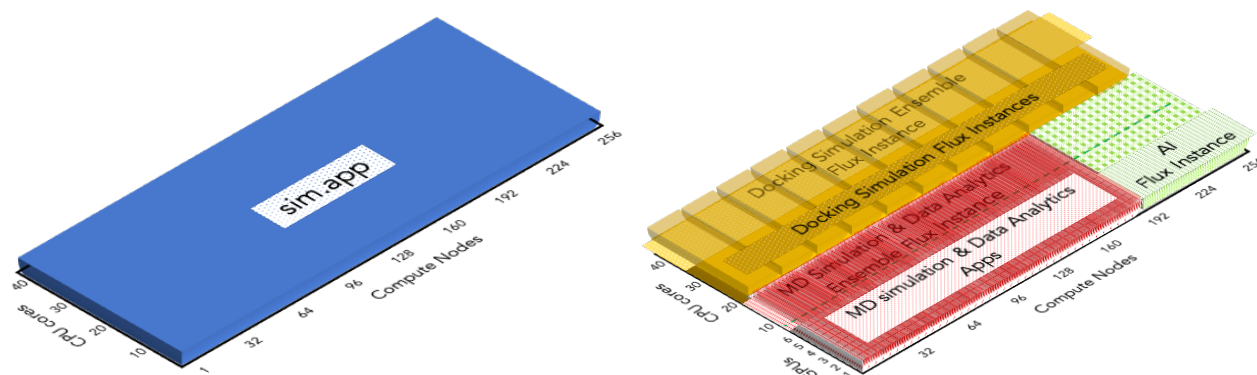
As part of the efforts in the ECP co-design Center for Efficient Exascale Discretizations (CEED), the MFEM team is also developing mathematical algorithms and software implementations for finite element methods that exploit increasing on-node concurrency targeting multiple complex architectures (e.g. GPUs). This work includes the libCEED low-level API library, the Laghos miniapp, and several other efforts available through CEED.

To reach its many customers and partners in NNSA, DOE SC, academia and industry, the MFEM team delivers regular releases on GitHub (e.g., mfem-4.2 in October 2020 and mfem-4.3 in July 2021) that include detailed documentation and many example codes. Code quality is ensured by smoke tests with GitHub actions on Linux, Mac, Windows and nightly regression testing at LLNL.

RAJA, Umpire, and CHAI RAJA, Umpire, and CHAI leverage the abstraction mechanisms available in modern C++ (C++11 and higher) compilers, such as Lambdas, policy templates, and constructor/destructor (RAII) patterns for resource management. They aim to provide performance portability at the *library* level, and they do not require special support from compilers. Targeting this level of the software stack gives DOE developers the flexibility to leverage standard parallel programming models like CUDA and OpenMP, without strictly depending on robust compiler support for these APIs. If necessary features are unavailable in compilers, library authors are not dependent on vendors for support, and they do not need to wait for these programming models to be fully implemented. These libraries allow applications to work correctly and perform well even if some functionality from OpenMP, CUDA, threading, etc. is missing.

Flux Flux combines fully hierarchical resource management with graph-based scheduling to improve the performance, portability, flexibility, and manageability of the scheduling and execution of both standard and complex scientific workflows on a wide range of HPC systems. Figure 97 visually contrasts the complexity of the conventional and new or emerging HPC paradigms in terms of how they utilize the resources allocated by the system’s workload manager. Nearly all existing products were designed when the workflows were much simpler as shown in Figure 97a. Yet, these solutions have significant difficulties with the emerging paradigm exemplified by Figure 97b, as well as with connecting and coordinating jobs. These problems have led users to develop their own ad hoc custom scheduling and resource management software or use tools that perform only workflow management or only scheduling. However, accomplishing sufficient job coordination without first-class support from a workload manager like Flux has already proven to be difficult for either approach. Perhaps more importantly, developing and maintaining ad hoc management software—especially in this era of ever-evolving HPC hardware architectures—has quickly become prohibitively expensive for supercomputing application teams. With Flux, a job script with multiple tasks submitted on a heterogeneous HPC system remains simple, requiring only a few more lines within the script.

AID Debugging a parallel code can be extremely difficult, and the most exhaustive approaches for finding errors can require a large amount of time to run. For example, understanding all of the potential interleavings of parallel threaded code requires combinatorial runtime with respect to the number of threads. It is not feasible to run this type of analysis at all times.



(a) Conventional Paradigm

(b) Emerging Paradigm

Figure 97: An illustration of how the computing resources allocated to a job—as granted by the HPC workload manager—differs between conventional and emerging scientific workflows. The notional X-axes depict the compute node IDs allocated to the job and Y-axes depict the IDs of computing resources in each of these nodes. (a) The conventional paradigm requires only a single parallel simulation application to run; (b) The emerging paradigm often requires many different types of tasks such as an ensemble of molecular dynamic (MD) parallel simulation applications and another ensemble of docking simulation applications along with in situ data analysis for the MD ensemble while these tasks are driven by an AI.

Our strategy is to provide a continuum of debugging tools – from the lightweight tools like STAT, which require only seconds to run and gives a high level overview of a code, to Archer, which requires lightweight code instrumentation, to replay-based fuzzing tools like ReMPI and FLiT, which run the code in a number of configurations to detect errors. With a suite of tools, we can enable developers to find the most common bugs quickly, while still being able to detect deep, hard-to-find issues given sufficient runtime and resources.

Caliper Caliper is implemented as a C++ library and is linked with applications. Application teams integrate it with their code by adding Caliper annotations at the application level. Contrast this with binary analysis and DWARF line mappings used by most performance tools, which are obtained automatically but increase tool complexity and are typically not linked with the application for regular runs.

Applications, their libraries, physics modules, and even runtime systems can be instrumented with Caliper and measured at the same time. All of these layers of the application stack provide additional context to Caliper measurements and enable deeper analysis of the relationships between different parts of the code.

Recent Progress Recent progress includes developments in Spack; MFEM; RAJA, Umpire, and CHAI; Flux; AID; and Caliper.

Spack Spack 0.16.0 was released and replaced Spack’s greedy concretizer algorithm with a much more aggressive solver based on Answer Set Programming. There was also a major overhaul to smooth developer workflows and allow working directly on checked-out code in Spack. Scalability of CI and binary cache creation in Spack was improved significantly (Figure 98). GPU support was hardened in Spack, and CUDA and ROCm package superclasses were updated.

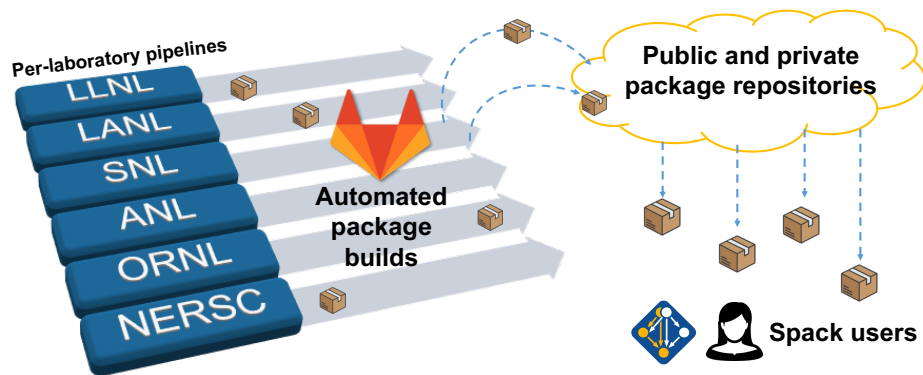


Figure 98: Spack build pipelines at facilities will provide HPC-native binary builds for users.

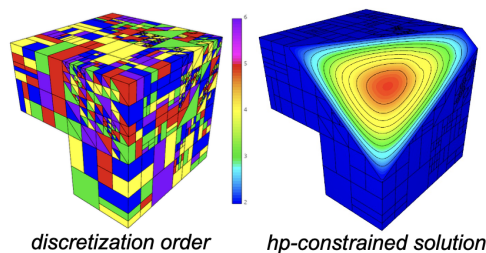


Figure 99: The MFEM team added support for variable order FEM spaces with general hp-refinement so higher order apps can better adapt to solutions.

MFEM Capabilities were added to MFEM for variable-order FEMs paces with general hp-refinement so HO apps can increase efficiency and better adapt to solutions (Figure 99). ALE discretization improvements were implemented for compressible flow in BLAST, including significant improvements in mesh optimization methods, dynamic adaptive mesh refinement (AMR), and improved transfer between high-order and low-order-refined simulations.

Machine	RAJA	CHAI	Umpire
Perlmutter	CUDA support production ready, actively used on Sierra. Continuing to investigate and improve performance.		
Frontier	Available in RAJA v0.11.0. Developed by AMD.	Available in CHAI v1.2.0. Developed by AMD.	Available in Umpire v1.0.0. Developed by AMD.
Aurora	Under development, also supported by ECP 2.3.1.18 RAJA/Kokkos. OpenMP 4.5 in RAJA already developed for Sierra.		Available in Umpire v4.0.0. Developed at Argonne.

Table 16: Status of RAJA, Umpire, and CHAI support for exascale platforms.

RAJA, Umpire, and CHAI Table 16 shows the status of RAJA, Umpire, and CHAI support for exascale platforms. The team updated support for AMD’s HIP programming model and hardened support for El Capitan COE machines. New versions of RAJA, Umpire, and CHAI with support for El Capitan COE machines were released. HIP support was added to continuous integration using GitLab at LLNL. Support for integration of Umpire into multiple ASC application codes continued.

Flux Flux was the basis for a workflow of ECP ExaAM (ExaConstit) in improving its throughput by 4×. The Flux-based ML drug design workflow was part of an SC20 COVID-19 Gordon Bell finalist submission.

Optimization of Flux enabled the MuMMI workflow to run across the full scale of Summit at ORNL. Support was added for exascale multitier storage in the Flux data-staging plugin and the Fluxion scheduler.

AID We demonstrated that our new correctness tools (FLiT and FPChecker) can analyze large LLNL code, thereby discovering previously unknown issues in LLNL code. We also facilitated tools co-design via the El Capitan tools working group.

Caliper We integrated the Caliper performance analysis tools into key LLNL applications. We also supported existing integration with MARBL via improvements to performance visualizations and analysis capabilities.

Next Steps Next steps for each effort have already been identified and are described below.

Spack In FY22, the team will work to further improve Spack developer workflows and reproducibility and expand on the new concretizer to handle compiler and runtime complexity in the exascale stack.

MFEM MFEM-based applications will be supported in their transition to exascale hardware. The team will continue to engage with ATDM application work, develop mini-apps, and provide support.

RAJA, Umpire, and CHAI GPU device capabilities will be expanded in Umpire, and the team will investigate the use of Umpire as a GPU offload allocator for RAJA. The team will continue to develop and deliver Umpire capabilities to applications in support of ASC milestone and user requirements, including interprocess shared memory to support shared on-node data and expanded functionality available inside GPU kernels. The team will also continue interactions with El Capitan Center of Excellence (COE) partners to resolve performance and correctness issues identified as application users begin testing on early access hardware.

Flux We will perform an end-to-end demonstration of HPE’s Rabbit-based multitiered storage scheduling and management. Support for Common Tools Interface (CTI)-based tool launching will be tuned to leverage HPE’s proprietary code-development tools—Valgrind4HPC and ATP. A Variorum-enabled power monitoring/capping subsystem, including power swing analysis, is also planned. Finally, the team will respond to the emerging needs of existing and new advanced workflow customers.

AID This effort will implement, evaluate, and harden a multilevel general-purpose GPU (GPGPU) debugging/code-correctness tool suite for early applications that run on Collaboration of Oak Ridge, Argonne, and Livermore (CORAL)-2 early access systems.

Caliper Integrations of SPOT and Caliper within LLNL applications will continue. Tools will be expanded with more GPU and MPI measurement capabilities, and support, porting, and deployment of Caliper will continue.

4.6.7 WBS 2.3.6.03 Sandia ATDM Software Technologies

Overview The Sandia ATDM Software Technologies projects are now aggregated to include Kokkos, Kokkos kernels, VTK-m, and Operating Systems and On-Node Runtime efforts.

The Kokkos programming model and C++ library enable performance portable on-compute-node parallelism for HPC/exascale C++ applications. Kokkos has been publicly available at <http://github.com/kokkos/kokkos> since May 2015 and is being used and evaluated by projects at DOE laboratories, PSAAP-II centers, other universities, and organizations such as DoD laboratories. Kokkos library implementation consists of a portable application programmer interface (API) and architecture specific back-ends, including OpenMP, Intel Xeon Phi, and CUDA on NVIDIA GPU. These back-ends are developed and optimized as new application-requested capabilities are added to Kokkos, back-end programming mechanisms evolve, and architectures change.

Kokkos Kernels implements on-node shared memory computational kernels for linear algebra and graph operations, using the Kokkos shared-memory parallel programming model. Kokkos Kernels forms the building blocks of a parallel linear algebra library like Tpetra in Trilinos that uses MPI and threads for parallelism, or it can be used stand-alone in ECP applications. Kokkos Kernels supports several Kokkos backends to support architectures like Intel CPUs, KNLs and NVIDIA GPUs. The algorithms and the implementations of the

performance-critical kernels in Kokkos Kernels are chosen carefully to match the features of the architectures. This allows ECP applications to utilize high performance kernels and transfers the burden to Kokkos Kernels developers to maintain them in future architectures. Kokkos Kernels also has support for calling vendor provided libraries where there are optimized kernels available.

VTK-m is a toolkit of scientific visualization algorithms for emerging processor architectures. VTK-m supports the fine-grained concurrency for data analysis and visualization algorithms required to drive extreme scale computing by providing abstract models for data and execution that can be applied to a variety of algorithms across many different processor architectures. The ECP/VTK-m project is building up the VTK-m codebase with the necessary visualization algorithm implementations that run across the varied hardware platforms to be leveraged at the exascale. We will be working with other ECP projects, such as ALPINE, to integrate the new VTK-m code into production software to enable visualization on our HPC systems. For the ASC/ATDM program, the VTK-m project will concentrate on support of ATDM applications and ASC's Advanced Technology Systems (ATS) as well as the ASTRA prototype system at Sandia. General information about VTK-m as well as source code can be found at <http://m.vtk.org>.

The OS and On-Node Runtime project focuses on the design, implementation, and evaluation of operating system and runtime system (OS/R) interfaces, mechanisms, and policies supporting the efficient execution of application codes on next-generation platforms. Priorities in this area include the development of lightweight tasking techniques that integrate network communication, interfaces between the runtime and OS for management of critical resources (including multi-level memory, non-volatile memory, and network interfaces), portable interfaces for managing power and energy, and resource isolation strategies at the operating system level that maintain scalability and performance while providing a more full-featured set of system services. The OS/R technologies developed by this project will be evaluated in the context of ATDM application codes running at large-scale on ASC platforms. Through close collaboration with vendors and the broader community, the intention is to drive the technologies developed by this project into vendor-supported system software stacks and gain wide adoption throughout the HPC community.

Key Challenges Each of the efforts described above faces key challenges to their successful implementation and deployment.

Kokkos The many-core revolution in computing is characterized by (1) a steady increase in the number of cores within individual computer chips; (2) a corresponding decrease in the amount of memory per core that must be shared by the cores of a chip, and (3) the diversity of computer chip architectures. This diversity is highly disruptive because each architecture imposes different complex and sometimes conflicting requirements on software to perform well on an architecture. Application software development teams are confronted with the dual challenges of: (1) inventing new parallel algorithms for many-core chips, (2) learning the different programming mechanisms of each architecture, and (3) creating and maintaining separate versions of their software specialized for each architecture. These tasks may involve considerable overhead for organizations in terms of time and cost. Adapting application software to changing HPC requirements is already becoming a large expense for HPC users and can be expected to grow as the diversity of HPC architectures continues to rise. An alternative, however, is creating software that is performance portable across current and future architectures.

Kokkos Kernels There are several challenges associated with the Kokkos Kernels work. Part of the complexity arises because profiling tools are not yet fully mature for advanced architectures and in this context profiling involves the interplay of several factors which require expert judgment to improve performance. Another challenging aspect is working on milestones that span a variety of projects and code bases. There is a strong dependence on the various application code development teams for our own team's success. In addition, we face a constant tension between the need for production ready tools and components in a realm where the state-of-the-art is still evolving.

VTK-m The scientific visualization research community has been building scalable HPC algorithms for over 15 years, and today there are multiple production tools that provide excellent scalability [298, 219]. That said, there are technology gaps in data analysis and visualization facing ATDM applications as they move to Exascale. As we approach Exascale, we find that we can rely less on disk storage systems as a holding area for all data between production (by the simulation) and consumption (by the visualization and analysis). To

circumvent this limitation, we must integrate our simulation and visualization into the same workflow and provide tools that allows us to run effectively and capture critical information.

OS & ONR Exascale challenges for system software span the areas of operating systems, networks, and run time systems. Container technologies are by now ubiquitous in the cloud computing space, but for HPC their immense potential has been limited by concerns about compatibility with security models and overhead costs. As vendors bring forward new network hardware for exascale, both vendors and application programmers lack insight into how applications actually use networks in practice, especially regarding the characteristics of the messages sent in production codes. As programming models like OpenMP at the node level and MPI at the inter-node level evolve, the particular needs of DOE applications must be addressed in both the development of standards and evaluation of provided run time system implementations.

Solution Strategy To mitigate these challenges, the team proposes the following actions for each effort.

Kokkos The Kokkos team developed a parallel programming model with flexible enough semantics that it can be mapped on a diverse set of HPC architectures including current multi-core CPUs and massively parallel GPUs. The programming model is implemented using C++ template abstractions, which allow a compile time translation to the underlying programming mechanism on each platform, using their respective primary tool chains (Figure 100). Compared to approaches which rely on source-to-source translators or special compilers, this way leverages the investment of vendors in their preferred programming mechanism without introducing additional, hard to maintain, tools in the compilation chain.

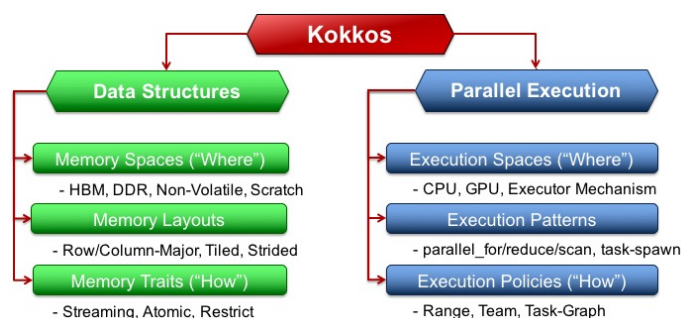


Figure 100: Kokkos Execution and Memory Abstractions

Kokkos Kernels The Kokkos Kernels team is taking a staged approach to profiling in regards to target architectures and the algorithms involved. We are also coordinating on a regular basis with the other projects that are involved in our work to minimize impediments. In response to the need for production ready tools, we are focusing on a hierarchical approach that involves producing robust, hardened code for core algorithms while simultaneously pursuing research ideas where appropriate.

VTK-m The VTK-m team is addressing its challenges through development of portable visualization algorithms for VTK-m and leveraging and expanding the Catalyst [219] *in situ* visualization library to apply this technology to ATDM applications on ASC platforms. VTK-m uses the notion of an abstract device adapter, which allows algorithms written once in VTK-m to run well on many computing architectures. The device adapter is constructed from a small but versatile set of data parallel primitives, which can be optimized for each platform [205]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [206, 207, 208].

OS & ONR The OS & ONR team is buying down risk for the use of containers by demonstrating exemplar application containerizations, e.g., for the ATDM SPARC application. We work with facilities staff to develop and implement strategies to deploy containers on our HPC systems and with dev-ops teams to ease the developer burden for code teams seeking to use containers. To better understand network resource utilization, we use an MPI simulator that accepts real network traces of application executions as inputs and provides

detailed analysis to inform network hardware vendors and application developers alike. We participate actively in both the OpenMP Language Committee and MPI Forum.

Recent Progress Recent progress includes developments in Kokkos, Kokkos Kernels, VTK-m, and OS and ONR.

Kokkos In FY21, the most significant accomplishment was the Kokkos 3.4 Release, the release that provides comprehensive support for exascale computing. This includes complete backends for AMD and Intel GPUs. The Kokkos team also added a Kokkos “Graph API” to help with latency limited execution cases. The feature allows the creation of a dependency graph object between multiple kernels, which can be repeatedly submitted for execution. Kokkos Graph can leverage the CUDA Graphs API introduced in CUDA 11. The Kokkos team also continued to provide high-quality support and consultation for ASC applications and libraries.

Kokkos Kernels In FY21, the most significant accomplishment was the Kokkos Kernels 3.4 Release, the release that provides significant kernel support for exascale computing. This includes math kernel support for AMD and Intel GPUs, leveraging the vendor math libraries where appropriate and providing native Kokkos implementations where necessary. Several improvements that are needed have also been identified in vendor math libraries. In addition, there have been several key algorithm improvements for GPU kernels:

- Developed a portable MIS-2 algorithm using Kokkos, which is 4-15x faster than state-of-the-art for GPUs. The new implementation has been run on NVIDIA/AMD GPUs and ARM/Intel CPUs. It has been integrated with the new algorithm with MueLu multigrid solver and should improve multigrid coarsening on GPUs.
- Improved Kokkos Kernels sparse matrix addition for sorted rows on GPU, now 50% faster overall than NVIDIA cuSPARSE.
- Optimized memory access patterns in Kokkos Kernels dense matrix-vector multiplication. In one Sierra-Thermal Fluids problem on ATS-2 system at LLNL. This sped up the overall radiosity solve by 15%.
- Rewrote Kokkos Kernels Symmetric Gauss Seidel to support sparse matrices with dense rows such as those appear in Sierra-Thermal Fluids problems. This impacts the ASC level 2 milestone results for Sierra-TF. The resulting hierarchical parallel algorithm for dense rows results in 6-8x speedup for Sierra-TF use cases.

The Kokkos Kernels team also continued to provide high-quality support and consultation for ASC applications and libraries.

VTK-m The VTK-m team made advances in three areas in FY21: support for performance evaluation of VTK-m, improvements to ParaView/Catalyst to enable *in situ* quantitative analysis, and a tighter integration of I/O and visualization capabilities to improve useability of the visualization tools. Early in FY21, the team created a driver program to exercise Catalyst and VTK-m without a large-scale application. The driver processes pre-computed data and has been used to evaluate GPU performance of VTK-m through Catalyst on Sandia’s Vortex system. Improvements to the Catalyst 5.9.0 API enable the injection of analysis capabilities without the need to recompile—reducing the dev-ops burden. In addition, Catalyst now has an “extract data” capability to support some of the planned quantitative-analysis capabilities in development for FY22. Finally, the team worked closely with Sandia’s production I/O team to better integrate visualization and I/O. These include improvements to the IOSS-Catalyst interface, an improved CGNS reader, and a new feature to IOSS that specifies a json-format script for embedded analysis.

OS & ONR The OS & ONR team had a number of recent accomplishments. We completed an in-depth performance analysis of a containerized version of the ATDM SPARC application at scale. We also completed an analysis of the impact of application perturbation from container infrastructure. Results show that containers running without contention for CPU resource have little impact on application perturbation; however, using Linux control groups to partition resources between concurrent containers can lead to noise-like events that have the potential to significantly degrade application performance. We also developed and

disseminated a Sandia ASC-wide container strategy that details how to integrate several activities supporting and leveraging container technologies. The strategy includes the creation of working groups, integration with existing DevOps activities, and further development efforts surrounding unprivileged container build solutions. We also led the addition of a new chapter to the MPI Standard on partitioned communication operations. This new functionality is aimed at improving the performance of network operations for highly multithreaded applications as well as more efficient methods of supporting communication from accelerators, such as GPUs and SmartNICs.

Next Steps Next steps for each effort have already been identified.

Kokkos With the comprehensive Kokkos exascale support in place, the focus of FY22 will be in helping the ASC libraries and application teams to obtain good performance on the early access AMD GPU machines in preparation for El Capitan. Specifically, the team will work with ASC applications and libraries to evaluate and improve performance on AMD GPUs. One specific effort will be working with the Trilinos Solvers team to improve their performance on AMD GPUs, which should help smooth the way for applications to El Capitan. The Kokkos team will also develop initial support for the Vanguard II system and identify future API/optimizations for Vanguard II backend. The Kokkos team will also continue to provide high-quality support and consultation for ASC applications and libraries. This will include providing Kokkos training to ASC customers if the available online recorded tutorials are insufficient and providing support via email, Slack, and Github to ASC customers.

Kokkos Kernels With the initial Kokkos Kernels exascale support in place, the focus of FY22 will be in providing complete kernel support (all needed sparse and dense linear algebra kernels, graph kernels) for the HIP backend (and AMD math library) for ASC libraries and application teams to obtain good performance on the early access AMD GPU machines in preparation for El Capitan. Specifically, the team will work with ASC applications and libraries to evaluate and improve kernel performance on AMD GPUs. Some specific efforts are as follows:

- Working with the Trilinos Solvers team to improve their performance of their component kernels on AMD GPUs.
- Providing BlockCrs support (SpMV, SpGEMM) and evaluate/optimize performance on CPUs and GPUs.
- Providing point ILU + SpTrsv performance improvements and block ILU+SpTrsv performance improvements (for Sierra-Thermal Fluids ASC L2 milestone).
- Providing team level batched ODE solvers (for Sierra-Thermal Fluids ASC L2 milestone).
- Ensuring performance of BlockDiag and BlockTridiag solvers on AMD architectures for El Cap readiness.

The Kokkos Kernels team will continue to engage the vendor community (Vanguard II vendor, NVIDIA, ARM, AMD) to develop and deliver kernels using Kokkos Kernels as reference implementation to better support ASC/ATDM codes and the broader CSE community. The Kokkos team also will continue to provide high-quality support and consultation for ASC applications and libraries. This will include providing support via email, Slack, and Github to ASC customers.

VTK-m The ATDM/VTK-m project will focus on two primary goals in FY22: evaluating and improving VTK-m performance on ATS platforms and providing *in situ* quantitative analysis capabilities in support of an ASC/ATDM Level II milestone on embedded capabilities. Addressing performance, the VTK-m team will work closely with the ECP VTK-m project to develop a performance evaluation suite and evaluate the Kokkos backend on the ASC ATS and/or Vanguard platforms and will deliver a report evaluating accelerated visualization algorithms on different platforms. Addressing quantitative analysis, the team will work closely with analysts to identify use-cases for Catalyst that include both visualization and quantitative outputs; then it will identify barriers to utility, useability, and adoption of the ParaView Python interface for the identified use cases. Results will be published in the FY22 ASC milestone report.

OS & ONR We plan to characterize how ASC/ATDM workloads use MPI communication in conjunction with GPUs to better understand workload performance and opportunities for optimization. We also plan to execute on the container infrastructure and support strategy developed last year, continuing to partner with vendors, facilities, and application/library developers to leverage container technologies for ASC/ATDM applications. We will also continue to contribute to the OpenMP and MPI standards bodies to shape the direction of the OpenMP and MPI parallel programming models to provided needed capabilities for ASC workloads.

Early Access System Experience Kokkos Kernels has been ported to both of the OLCF early-access systems Spock and Crusher. All tests pass on Crusher. We have initial performance results for sparse kernels such as sparse matrix-vector multiplication on both Spock and Crusher. We are working on additional performance optimizations. Several compiler/runtime issues were identified in this porting process. These issues were reported to the appropriate teams, and are fixed now.

The VTK-m team is heavily using the OLCF Spock system and has identified issues that require the use of ROCM 5 which hasn't been released. The VTK-m team was also recently granted access to Crusher system, but due to funding delays has not been able to make progress porting or evaluating VTK-m on that platform.

5. CONCLUSION

ECP ST is providing a collection of essential software capabilities necessary for successful results from Exascale computing platforms, while also delivery a suite of products that can be sustained into the future. This Capabilities Assessment Report and subsequent versions will provide a periodic summary of capabilities, plans, and challenges as the ECP proceeds.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable Exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation’s Exascale computing imperative.

REFERENCES

- [1] Todd Gamblin, Matthew P. LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and W. Scott Futral. The Spack Package Manager: Bringing order to HPC software chaos. In *Supercomputing 2015 (SC'15)*, Austin, Texas, November 15-20 2015. LLNL-CONF-669890.
- [2] Todd Gamblin, Gregory Becker, Peter Scheibel, Matt Legendre, and Mario Melara. Managing HPC Software Complexity with Spack. In *Exascale Computing Project 2nd Annual Meeting*, Knoxville, TN, February 6-8 2018. Half day.
- [3] xSDK Community Policies Web page. <http://xsdk.info/policies>.
- [4] M. Ben Olson, Brandon Kammerdiener, Kshitij A. Doshi, Terry Jones, and Michael R. Jantz. Online application guidance for heterogeneous memory systems, 2021.
- [5] Paul K Romano and Benoit Forget. The openmc monte carlo particle transport code. *Annals of Nuclear Energy*, 51:274–281, 2013.
- [6] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. XSbench - the development and verification of a performance abstraction for Monte Carlo reactor analysis. In *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, Kyoto, 2014.
- [7] John R. Tramm, Andrew R. Siegel, Benoit Forget, and Colin Josey. Performance analysis of a reduced data movement algorithm for neutron cross section data in monte carlo simulations. In *EASC 2014 - Solving Software Challenges for Exascale*, Stockholm, 2014.
- [8] David McCallen, Houjun Tang, Suiwen Wu, Eric Eckert, Junfei Huang, and N Anders Petersson. Coupling of regional geophysics and local soil-structure models in the EQSIM fault-to-structure earthquake simulation framework. *The International Journal of High Performance Computing Applications*, pages 1–15, 2021.
- [9] Trilinos Web page. <https://trilinos.org>.
- [10] LLVM Compiler Infrastructure. LLVM Compiler Infrastructure. <http://www.llvm.org>.
- [11] Supercontainers Presentation. <https://oaciss.uoregon.edu/E4S-Forum19/talks/Younger-E4S.pdf>.
- [12] Paul Basco. DOE Order 413.3B: Program and Project Management (PM) for the Acquisition of Capital Assets, Significant Changes to the Order. https://www.energy.gov/sites/prod/files/maprod/documents/15-1025_Bosco.pdf.
- [13] E4S Community Policies. <https://e4s-project.github.io/policies.html>.
- [14] Michael A. Heroux. Episode 17: Making the development of scientific applications effective and efficient. <https://soundcloud.com/exascale-computing-project/episode-17-making-the-development-of-scientific-applications-effective-and-efficient>.
- [15] Livermore Computing. Toss: Speeding up commodity cluster computing. <https://computation.llnl.gov/projects/toss-speeding-commodity-cluster-computing>.
- [16] OpenHPC. Community building blocks for hpc systems. <http://openhpc.community>.
- [17] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. Fine-grained multithreading support for hybrid threaded MPI programming. *Int. J. High Perform. Comput. Appl.*, 24(1):49–57, February 2010.
- [18] Rajeev Thakur and William Gropp. Test suite for evaluating performance of multithreaded MPI communication. *Parallel Comput.*, 35(12):608–617, December 2009.
- [19] William Gropp and Ewing Lusk. Fault tolerance in message passing interface programs. *The International Journal of High Performance Computing Applications*, 18(3):363–372, 2004.

- [20] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur, and J. L. Traeff. MPI on Millions of Cores. *Parallel Processing Letters (PPL)*, 21(1):45–60, March 2011.
- [21] D. Buntinas, B. Goglin, D. Goodell, G. Mercier, and S. Moreaud. Cache-efficient, intranode, large-message MPI communication with MPICH2-nemesis. In *2009 International Conference on Parallel Processing*, pages 462–469, September 2009.
- [22] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur and J. L. Traeff. MPI on millions of cores. *Parallel Processing Letters*, 21(1):45–60, 2011.
- [23] Y. Guo, C. J. Archer, M. Blocksome, S. Parker, W. Bland, K. Raffanetti, and P. Balaji. Memory compression techniques for network address management in MPI. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1008–1017, May 2017.
- [24] Nikela Papadopoulou, Lena Oden, and Pavan Balaji. A performance study of UCX over infiniband. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid '17*, pages 345–354, Piscataway, NJ, USA, 2017. IEEE Press.
- [25] Ken Raffanetti, Abdelhalim Amer, Lena Oden, Charles Archer, Wesley Bland, Hajime Fujita, Yanfei Guo, Tomislav Janjusic, Dmitry Durnov, Michael Blocksome, Min Si, Sangmin Seo, Akhil Langer, Gengbin Zheng, Masamichi Takagi, Paul Coffman, Jithin Jose, Sayantan Sur, Alexander Sannikov, Sergey Oblomov, Michael Chuvelev, Masayuki Hatanaka, Xin Zhao, Paul Fischer, Thilina Rathnayake, Matt Otten, Misun Min, and Pavan Balaji. Why is MPI so slow?: Analyzing the fundamental limits in implementing MPI-3.1. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17*, pages 62:1–62:12, New York, NY, USA, 2017. ACM.
- [26] Ashwin M. Aji, Lokendra S. Panwar, Feng Ji, Karthik Murthy, Milind Chabbi, Pavan Balaji, Keith R. Bisset, James Dinan, Wu chun Feng, John Mellor-Crummey, Xiaosong Ma, and Rajeev Thakur. MPI-ACC: Accelerator-aware MPI for scientific applications. *IEEE Trans. Parallel Distrib. Syst.*, 27(5):1401–1414, May 2016.
- [27] Humayun Arafat, James Dinan, Sriram Krishnamoorthy, Pavan Balaji, and P. Sadayappan. Work stealing for GPU-accelerated parallel programs in a global address space framework. *Concurr. Comput. : Pract. Exper.*, 28(13):3637–3654, September 2016.
- [28] F. Ji, J. S. Dinan, D. T. Buntinas, P. Balaji, X. Ma and W. chun Feng. Optimizing GPU-to-GPU intranode communication in MPI. In *2012 International Workshop on Accelerators and Hybrid Exascale Systems, ASHES 12*, 2012.
- [29] Lena Oden and Pavan Balaji. Hexe: A toolkit for heterogeneous memory management. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2017.
- [30] Giuseppe Congiu and Pavan Balaji. Evaluating the impact of high-bandwidth memory on mpi communications. In *IEEE International Conference on Computer and Communications*, 2018.
- [31] Mohammad Javad Rashti, Jonathan Green, Pavan Balaji, Ahmad Afsahi, and William Gropp. Multi-core and network aware MPI topology functions. In Yiannis Cotronis, Anthony Danalis, Dimitrios S. Nikolopoulos, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface*, pages 50–60, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [32] Torsten Hoefler, Rolf Rabenseifner, Hubert Ritzdorf, Bronis R. de Supinski, Rajeev Thakur, and Jesper Larsson Traff. The scalable process topology interface of MPI 2.2. *Concurr. Comput. : Pract. Exper.*, 23(4):293–310, March 2011.
- [33] Leonardo Arturo Bautista Gomez Robert Latham and Pavan Balaji. Portable topology-aware MPI-I/O. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2017.
- [34] Min Si and Pavan Balaji. Process-based asynchronous progress model for MPI point-to-point communication. In *IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2017.

- [35] Pavan Balaji Seyed Hessamedin Mirsadeghi, Jesper Larsson Traff and Ahmad Afsahi. Exploiting common neighborhoods to optimize MPI neighborhood collectives. In *IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2017.
- [36] Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, SC '06*, New York, NY, USA, 2006. ACM.
- [37] H. V. Dang, S. Seo, A. Amer, and P. Balaji. Advanced thread synchronization for multithreaded MPI implementations. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 314–324, May 2017.
- [38] Abdelhalim Amer, Charles Archer, Michael Blocksome, Chongxiao Cao, Michael Chuvelev, Hajime Fujita, Maria Garzaran, Yanfei Guo, Jeff R. Hammond, Shintaro Iwasaki, Kenneth J. Raffanetti, Mikhail Shiryaev, Min Si, Kenjiro Taura, Sagar Thapaliya, and Pavan Balaji. Software Combining to Mitigate Multithreaded MPI Contention. In *Proceedings of the ACM International Conference on Supercomputing, ICS '19*, pages 367–379, New York, NY, USA, 2019. ACM.
- [39] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. Carns, A. Castello, D. Genet, T. Herault, S. Iwasaki, P. Jindal, L. V. Kale, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, and P. Beckman. Argobots: A lightweight low-level threading and tasking framework. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):512–526, March 2018.
- [40] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, Sangmin Seo, and Pavan Balaji. BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads. In *Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques, PACT '19*, New York, NY, USA, 2019. ACM.
- [41] Rohit Zambre, Aparna Chandramowliswharan, and Pavan Balaji. How I learned to stop worrying about user-visible endpoints and love MPI. In *Proceedings of the 34th ACM International Conference on Supercomputing, ICS '20*, New York, NY, USA, 2020. ACM.
- [42] Q. Cao, Y. Pei, T. Herault, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. E. Keyes, and J. Dongarra. Performance Analysis of Tile Low-Rank Cholesky Factorization Using ParSEC Instrumentation Tools. In *ProTools'19, ProTools'19*, 2019.
- [43] Y. Pei, G. Bosilca, I. Yamazaki, A. Ida, and J. Dongarra. Evaluation of Programming Models to Address Load Imbalance on Distributed Multi-Core CPUs: A Case Study with Block Low-Rank Factorization. In *Parallel Applications Workshop, Alternatives To MPI+X, PAW-ATM 2019*, Nov 2019.
- [44] Thomas Herault, Yves Robert, George Bosilca, and Jack J. Dongarra. Generic matrix multiplication for multi-GPU accelerated distributed-memory platforms over ParSEC. *Scala19*, 2019.
- [45] Thomas Herault, Yves Robert, George Bosilca, Robert J Harrison, Cannada A Lewis, Edward F Valeev, and Jack J Dongarra. Distributed-memory multi-GPU block-sparse tensor contraction for electronic structure (revised version). Research Report RR-9365, Inria - Research Centre Grenoble – Rhône-Alpes, October 2020.
- [46] Ole Schütt, Peter Messmer, Jürg Hutter, and Joost VandeVondele. *GPU-Accelerated Sparse Matrix-Matrix Multiplication for Linear Scaling Density Functional Theory*, chapter 8, pages 173–190. John Wiley & Sons, Ltd, 2016.
- [47] Chong Peng, Justus A Calvin, Fabijan Pavošević, Jinmei Zhang, and Edward F Valeev. Massively Parallel Implementation of Explicitly Correlated Coupled-Cluster Singles and Doubles Using TiledArray Framework. *J. Phys. Chem. A*, 120(51):10231–10244, December 2016.
- [48] GASNet website. <https://gasnet.lbl.gov/>.

- [49] John Bachan, Scott B. Baden, Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, Dan Bonachea, Paul H. Hargrove, and Hadia Ahmed. UPC++: A High-Performance Communication Framework for Asynchronous Computation. In *Proceedings of the 33rd IEEE International Parallel & Distributed Processing Symposium*, IPDPS. IEEE, 2019. <https://doi.org/10.25344/S4V88H>.
- [50] UPC++ website. <https://upcxx.lbl.gov/>.
- [51] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: expressing locality and independence with logical regions. In *Proceedings of the international conference on high performance computing, networking, storage and analysis*, page 66. IEEE Computer Society Press, 2012.
- [52] The Legion Programming System website. <http://legion.stanford.edu/>.
- [53] Bradford L. Chamberlain. Chapel. In *Programming Models for Parallel Computing*. The MIT Press, 2015.
- [54] The Chapel Parallel Programming Language website. <https://chapel-lang.org/>.
- [55] Dan Bonachea and Paul H. Hargrove. GASNet-EX: A High-Performance, Portable Communication Library for Exascale. In *Proceedings of Languages and Compilers for Parallel Computing (LCPC'18)*, volume 11882 of *Lecture Notes in Computer Science*. Springer International Publishing, October 2018. Lawrence Berkeley National Laboratory Technical Report (LBNL-2001174). <https://doi.org/10.25344/S4QP4W>.
- [56] Dan Bonachea and Paul Hargrove. GASNet specification, v1.8.1. Technical Report LBNL-2001064, Lawrence Berkeley National Laboratory, August 2017. <https://doi.org/10.2172/1398512>.
- [57] Paul H. Hargrove, Dan Bonachea, Colin A. MacLean, and Daniel Waters. GASNet-EX Memory Kinds: Support for Device Memory in PGAS Programming Models (poster). November 2021. Research Poster at SC21. <https://doi.org/10.25344/S4P306>.
- [58] John Bachan, Scott B. Baden, Dan Bonachea, Max Grossman, Paul H. Hargrove, Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, Brian van Straalen, and Daniel Waters. UPC++ v1.0 Programmer's Guide, Revision 2021.9.0. Technical Report LBNL-2001424, Lawrence Berkeley National Laboratory, September 2021. <https://doi.org/10.25344/S4SW2T>.
- [59] Dan Bonachea and Amir Kamil. UPC++ v1.0 Specification, Revision 2021.9.0. Technical Report LBNL-2001425, Lawrence Berkeley National Laboratory, September 2021. <https://doi.org/10.25344/S4XK53>.
- [60] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick. UPC++: A PGAS extension for C++. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1105–1114, May 2014. <https://doi.org/10.1109/IPDPS.2014.115>.
- [61] Daniel Waters, Colin A. MacLean, Dan Bonachea, and Paul H. Hargrove. Demonstrating UPC++/Kokkos interoperability in a heat conduction simulation (extended abstract). In *2021 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI+X (PAW-ATM)*, November 2021. <https://doi.org/10.25344/S4630V>.
- [62] Katherine A. Yelick, Amir Kamil, Damian Rouson, Dan Bonachea, and Paul H. Hargrove. UPC++: An asynchronous RMA/RPC library for distributed C++ applications. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC21)*, November 2021. Half-day tutorial at SC21. <https://go.lbl.gov/sc21>.
- [63] Amir Kamil and Dan Bonachea. Optimization of asynchronous communication operations through eager notifications. In *2021 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI+X (PAW-ATM)*, November 2021. <https://doi.org/10.25344/S42C71>.

- [64] David E. Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E. Grant, Thomas Naughton, Howard P. Pritchard, Martin Schulz, and Geoffroy R. Vallee. A survey of MPI usage in the U.S. Exascale Computing Program. Technical Report ORNL/SPR-2018/790, Oak Ridge National Laboratory, 2018. <https://doi.org/10.2172/1462877>.
- [65] Swann Perarnau, Judicael A Zounmevo, Matthieu Dreher, Brian C Van Essen, Roberto Gioiosa, Kamil Iskra, Maya B Gokhale, Kazutomo Yoshii, and Pete Beckman. Argo NodeOS: Toward unified resource management for exascale. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 153–162. IEEE, 2017.
- [66] J. S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, T. Humble, C. Schuman, B. Van Essen, S. Yoo, A. Aiken, D. Bernholdt, S. Byna, K. Cameron, F. Cappello, B. Chapman, A. Chien, M. Hall, R. Hartman-Baker, Z. Lan, M. Lang, J. Leidel, S. Li, R. Lucas, J. Mellor-Crummey, P. Peltz Jr., T. Peterka, M. Strout, and J. Wilke. Extreme heterogeneity 2018 - productive computational science in the era of extreme heterogeneity: Report for DOE ASCR workshop on extreme heterogeneity. Technical report, USDOE Office of Science (SC) (United States), 2018.
- [67] J.E. Denny, S. Lee, and J.S. Vetter. Clacc: Translating OpenACC to OpenMP in Clang. In *2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, Dallas, TX, USA, 2018. IEEE.
- [68] Johannes Doerfert, Joseph Huber, Stefan Stipanovic, Giorgis Georgakoudis, Hamilton Tobon Mosquera, and Shilei Tian. (OpenMP) Parallelism Aware Optimizations. https://whova.com/embedded/session/llvm_202010/1162344/, 2020.
- [69] Giorgis Georgakoudis, Johannes Doerfert, Ignacio Laguna, and Thomas R. W. Scogland. FAROS: A Framework to Analyze OpenMP Compilation Through Benchmarking and Compiler Optimization Analysis. In Kent Milfeld, Bronis R. de Supinski, Lars Koesterke, and Jannis Klinkenberg, editors, *OpenMP: Portable Multi-Level Parallelism on Modern Systems*, pages 3–17, Cham, 2020. Springer International Publishing.
- [70] Michael Kruse. Using OpenMP loop transformations with Clang. OpenMP Booth Talk at SC21, 2020.
- [71] Michael Kruse. Loop transformations using clang’s abstract syntax tree. In *50th International Conference on Parallel Processing Workshop, ICPP Workshops ’21*, New York, NY, USA, 2021. Association for Computing Machinery.
- [72] Michael Wu, Xingfu abd Kruse, Prasanna Balaprakash, Hal Finkel, Paul Hovland, and Valerie Taylor. Customized Monte Carlo Tree Search for LLVM/Polly’s Composable Loop Optimization Transformations. *PMBS Workshop @SC21*, 2021.
- [73] Hal Finkel, Alex McCaskey, Tobi Popoola, Dmitry Lyakh, and Johannes Doerfert. *Really* Embedding Domain-Specific Languages into C++. *LLVM-HPC Workshop @SC*, 2020.
- [74] OpenACC: Commerical Compilers. [Online]. Available: <http://openacc.org/tools>.
- [75] Kyle Friedline, Sunita Chandrasekaran, M. Graham Lopez, and Oscar Hernandez. OpenACC 2.5 Validation Testsuite Targeting Multiple Architectures. In *High Performance Computing*, pages 557–575, Cham, 2017. Springer International Publishing.
- [76] SPEC ACCEL. [Online]. Available: <https://www.spec.org/accel/>.
- [77] Michael Kruse, Hal Finkel, and Xingfu Wu. Autotuning Search Space for Loop Transformations. *LLVM-HPC Workshop @SC*, 2020.
- [78] Tuowen Zhao, Samuel Williams, Mary Hall, and Hans Johansen. Delivering performance portable stencil computations on cpus and gpus using bricks. In *Proceedings of the International Workshop on Performance, Portability and Productivity in HPC (P3HPC), SC’18*, Nov 2018.

- [79] T. Zhao, P. Basu, S. Williams, M. Hall, and H. Johansen. Exploiting reuse and vectorization in blocked stencil computations on CPUs and GPUs. In *accepted and to appear, SC 2019*, Nov 2019.
- [80] T. Zhao, S. Williams, M. Hall, and H. Johansen. Pack-free stencil ghost zone exchange. In *submitted to IPDPS '20*, 2019.
- [81] GWT-TUD GmbH. Vampir. <https://vampir.eu>, 2020. Accessed: 2020-10-15.
- [82] Jungwon Kim, Kittisak Sajjapongse, Seyong Lee, and Jeffrey S. Vetter. Design and Implementation of Papyrus: Parallel Aggregate Persistent Storage. In *Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium*, IPDPS '17, pages 1151–1162, 2017.
- [83] Jungwon Kim, Seyong Lee, and Jeffrey S. Vetter. PapyrusKV: A high-performance parallel key-value store for distributed NVM architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 57:1–57:14, 2017.
- [84] Jungwon Kim Kim and Jeffrey S. Vetter. Implementing Efficient Data Compression and Encryption in a Persistent Key-value Store for HPC. *The International Journal of High Performance Computing Applications*, 33(6):1098–1112, 2019.
- [85] Zheming Jin and Jeffrey Vetter. Evaluating cuda portability with hipcl and dpct. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 371–376, 2021.
- [86] Zheming Jin and Jeffrey Vetter. Evaluating the performance of integer sum reduction in sycl on gpus. In *50th International Conference on Parallel Processing Workshop, ICPP Workshops '21*, New York, NY, USA, 2021. Association for Computing Machinery.
- [87] M. Thavappiragasam, V. Kale, O. Hernandez, and A. Sedova. Addressing load imbalance in bioinformatics and biomedical applications: Efficient scheduling across multiple gpus. In *2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1992–1999, Los Alamitos, CA, USA, dec 2021. IEEE Computer Society.
- [88] Bronis R. De Supinski and Michael Klemm. OpenMP Technical Report 8: Version 5.0 Preview 2. <http://www.openmp.org/wp-content/uploads/openmp-tr8.pdf>, 2017. [Online; accessed 31-November-2019].
- [89] Oleksandr Zinenko, Sven Verdoolaege, Chandan Reddy, Jun Shirako, Tobias Grosser, Vivek Sarkar, and Albert Cohen. Modeling the conflicting demands of parallelism and temporal/spatial locality in affine scheduling. In *Proceedings of the 27th International Conference on Compiler Construction*, pages 3–13. ACM, 2018.
- [90] Jose Manuel Monsalve Diaz, Kyle Friedline, Swaroop Pophale, Oscar Hernandez, David Bernholdt, and Sunita Chandrasekaran. Analysis of OpenMP 4.5 Offloading in Implementations: Correctness and Overhead. *Parallel Computing*, page 102546, 08 2019.
- [91] Johannes Doerfert, Jose Diaz, and Hal Finkel. *The TRegion Interface and Compiler Optimizations for OpenMP Target Regions*, pages 153–167. 08 2019.
- [92] Alok Mishra, Martin Kong, and Barbara Chapman. Kernel Fusion/Decomposition for Automatic GPU-offloading. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, pages 283–284, Piscataway, NJ, USA, 2019. IEEE Press.
- [93] Lingda Li and Barbara Chapman. Compiler Assisted Hybrid Implicit and Explicit GPU Memory Management under Unified Address Space. In *Proceedings of the 31st ACM/IEE International Conference for High Performance Computing, Networking, Storage, and Analysis in Denver, CO, USA (SC '19)*, November 2019.

- [94] Michael Kruse and Hal Finkel. Design and Use of Loop Transformation Pragmas. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, 2019.
- [95] Vinu Sreenivasan, Rajath Javali, Mary W. Hall, Prasanna Balaprakash, Thomas R. W. Scogland, and Bronis R. de Supinski. A Framework for Enabling OpenMP Autotuning. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 50–60, 2019.
- [96] V Sreenivasan, R Javali, M Hall, P Balaprakash, and B R de Supinski. A Framework for Enabling OpenMP Autotuning. 11718, 6 2019.
- [97] Thomas R. W. Scogland, Dan Sunderland, Stephen L. Olivier, David S. Hollman, Noah Evans, and Bronis R. de Supinski. Making OpenMP Ready for C++ Executors. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 320–332, 2019.
- [98] Yonghong Yan, Anjia Wang, Chunhua Liao, Thomas R. W. Scogland, and Bronis R. de Supinski. Extending OpenMP Metadirective Semantics for Runtime Adaptation. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 201–214, 2019.
- [99] Vivek Kale, Christian Iwainsky, Michael Klemm, Jonas H. Müller Korndörfer, and Florina M. Ciorba. Toward a Standard Interface for User-Defined Scheduling in OpenMP. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 186–200, 2019.
- [100] Seonmyeong Bak, Yanfei Guo, Pavan Balaji, and Vivek Sarkar. Optimized Execution of Parallel Loops via User-Defined Scheduling Policies. In *ICPP*, 2019.
- [101] Jonas H. Muller Kondorfer, Florina Ciorba, Christian Iwainsky, Johannes Doerfert, Hal Finkel, Vivek Kale, and Michael Klemm. A Runtime Approach for Dynamic Load Balancing of OpenMP Parallel Loops in LLVM. In *Proceedings of the 31st ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis in Denver, CO, USA (SC '19)*, November 2019.
- [102] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, Sangmin Seo, and Pavan Balaji. BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads. In *Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques (PACT '19)*, September 2019.
- [103] Alok Mishra, Abid M. Malik, and Barbara Chapman. Using Machine Learning for OpenMP GPU Offloading in LLVM. In *ACM SRC to be held at SC20*.
- [104] Alok Mishra, Lingda Li, Martin Kong, Hal Finkel, and Barbara Chapman. Benchmarking and Evaluating Unified Memory for OpenMP GPU Offloading. In *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC, LLVM-HPC@SC 2017, Denver, CO, USA, November 13, 2017*, pages 6:1–6:10, 2017.
- [105] Seonmyeong Bak, Colleen Bertoni, Swen Boehm, Reuben Budiardja, Barbara Chapman, Johannes Doerfert, Markus Eisenbach, Hal Finkel, Oscar Hernandez, Joseph Huber, Shintaro Iwasaki, Vivek Kale, Paul Kent, JaeHyuk Kwack, Meifeng Lin, Piotr Luszczek, Ye Luo, Buu Pham, Swaroop Pophale, Kiran Ravikumar, Vivek Sarkar, Thomas Scogland, Shilei Tian, and P.K. Yeung. OpenMP Application Experiences: Porting to Accelerated Nodes. In *Journal of Parallel Computing*, October 2021.
- [106] Ralf S. Engelschall. GNU portable threads (Pth). <http://www.gnu.org/software/pth>, 1999.
- [107] K. Taura and Akinori Yonezawa. Fine-grain multithreading with minimal compiler support – a cost effective approach to implementing efficient multithreading languages. In *PLDI*, pages 320–333, 1997.
- [108] S. Thibault. A flexible thread scheduler for hierarchical multiprocessor machines. In *COSET*, 2005.

- [109] J. Nakashima and Kenjiro Taura. MassiveThreads: A thread library for high productivity languages. In *Concurrent Objects and Beyond*, pages 222–238. Springer, 2014.
- [110] K. B. Wheeler, Richard C. Murphy, and Douglas Thain. Qthreads: An API for programming with millions of lightweight threads. In *MTAAP*, 2008.
- [111] K. Taura, Kunio Tabata, and Akinori Yonezawa. StackThreads/MP: Integrating futures into calling standards. In *PPPoP*, pages 60–71, 1999.
- [112] A. Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *SenSys*, pages 29–42, 2006.
- [113] Chuck Pheatt. Intel® threading building blocks. *Journal of Computing Sciences in Colleges*, 23(4):298–298, 2008.
- [114] M. Pérache, Hervé Jourden, and Raymond Namyst. MPC: A unified parallel runtime for clusters of NUMA machines. In *EuroPar*, pages 78–88, 2008.
- [115] A. Adya, Jon Howell, Marvin Theimer, William J. Bolosky, and John R. Douceur. Cooperative task management without manual stack management. In *ATC*, 2002.
- [116] CORPORATE SunSoft. *Solaris multithreaded programming guide*. Prentice-Hall, Inc., 1995.
- [117] R. von Behren, Jeremy Condit, Feng Zhou, George C. Necula, and Eric Brewer. Capriccio: Scalable threads for internet services. In *SOSP*, pages 268–281, 2003.
- [118] G. Shekhtman and Mike Abbott. State threads library for internet applications. <http://state-threads.sourceforge.net/>, 2009.
- [119] P. Li and Steve Zdancewic. Combining events and threads for scalable network services implementation and evaluation of monadic, application-level concurrency primitives. In *PLDI*, pages 189–199, 2007.
- [120] A. Porterfield, Nassib Nassar, and Rob Fowler. Multi-threaded library for many-core systems. In *MTAAP*, 2009.
- [121] J. del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. TiNy threads: A thread virtual machine for the Cyclops64 cellular architecture. In *WMPP*, 2005.
- [122] Intel OpenMP runtime library. <https://www.openmpRTL.org/>, 2016.
- [123] Barcelona Supercomputing Center. Nanos++. <https://pm.bsc.es/projects/nanox/>, 2016.
- [124] L. V. Kalé, Josh Yelon, and T. Knuff. Threads for interoperable parallel programming. In *LCPC*, pages 534–552, 1996.
- [125] S. Treichler, Michael Bauer, and Alex Aiken. Realm: An event-based low-level runtime for distributed memory architectures. In *PACT*, pages 263–276, 2014.
- [126] Sangmin Seo, Abdelhalim Amer, Pavan Balaji, Cyril Bordage, George Bosilca, Alex Brooks, Philip Carns, Adrián Castelló, Damien Genet, Thomas Herault, et al. Argobots: a lightweight low-level threading and tasking framework. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):512–526, 2018.
- [127] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, and Pavan Balaji. Lessons learned from analyzing dynamic promotion for user-level threading. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 23. IEEE Press, 2018.
- [128] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, and Pavan Balaji. Analyzing the performance trade-off in implementing user-level threads. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1859–1877, 2020.

- [129] Abdelhalim Amer Amer, Milind Chabbi, Huiwei Lu, Yanji Wei, Jeff Hammond, Satoshi Matsuoka, and Pavan Balaji. Lock contention management in multithreaded mpi. *ACM Transactions on Parallel Computing*, 2018.
- [130] H. V. Dang, S. Seo, A. Amer, and P. Balaji. Advanced thread synchronization for multithreaded mpi implementations. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 314–324, May 2017.
- [131] Abdelhalim Amer, Huiwei Lu, Yanjie Wei, Pavan Balaji, and Satoshi Matsuoka. Mpi+threads: Runtime contention and remedies. *SIGPLAN Not.*, 50(8):239–248, January 2015.
- [132] Shilei Tian, Johannes Doerfert, and Barbara Chapman. Concurrent Execution of Deferred OpenMP Target Tasks with Hidden Helper Threads. In *In Workshop on Languages and Compilers for Parallel Computing (LCPC 2020)*.
- [133] Shilei Tian, Jon Chesterfield, Johannes Doerfert, and Barbara M. Chapman. Experience report: Writing a portable GPU runtime with openmp 5.1. In Simon McIntosh-Smith, Bronis R. de Supinski, and Jannis Klinkenberg, editors, *OpenMP: Enabling Massive Node-Level Parallelism - 17th International Workshop on OpenMP, IWOMP 2021, Bristol, UK, September 14-16, 2021, Proceedings*, volume 12870 of *Lecture Notes in Computer Science*, pages 159–169. Springer, 2021.
- [134] Atmn Patel, Shilei Tian, Johannes Doerfert, and Barbara M. Chapman. A Virtual GPU as Developer-Friendly OpenMP Offload Target. In *ICPP*, 2021.
- [135] Johannes Doerfert, Joseph Huber, and Melanie Cornelius. Advancing openmp offload debugging capabilities in LLVM. In Federico Silla and Osni Marques, editors, *ICPP Workshops 2021: 50th International Conference on Parallel Processing, Virtual Event / Lemont (near Chicago), IL, USA, August 9-12, 2021*, pages 20:1–20:8. ACM, 2021.
- [136] Joseph Huber, Weile Wei, Giorgis Georgakoudis, Johannes Doerfert, and Oscar R. Hernandez. A case study of llvm-based analysis for optimizing SIMD code generation. In Simon McIntosh-Smith, Bronis R. de Supinski, and Jannis Klinkenberg, editors, *OpenMP: Enabling Massive Node-Level Parallelism - 17th International Workshop on OpenMP, IWOMP 2021, Bristol, UK, September 14-16, 2021, Proceedings*, volume 12870 of *Lecture Notes in Computer Science*, pages 142–155. Springer, 2021.
- [137] J. Huber, M. Cornelius, G. Georgakoudis, S. Tian, J. Diaz, K. Dinel, B. Chapman, J. Doerfert. Efficient Execution of OpenMP on GPUs. In *Code Generation and Optimizations (CGO)*, 2022.
- [138] J. Doerfert, A. Patel, S. Tian, J. Huber, J. Diaz, B. Chapman, G. Georgakoudis. Co-Designing an OpenMP GPU Runtime and Optimizations for Near-Zero Overhead Execution. In *International Parallel Distributed Processing Symposium (IPDPS)*, 2022.
- [139] Xinmin Tian. An update on the intel compiler. In *OpenMPCon*, 2021.
- [140] Scott LeGrand, Aaron Scheinberg, Andreas F Tillack, Mathialakan Thavappiragasam, Josh V Vermaas, Rupesh Agarwal, Jeff Larkin, Duncan Poole, Diogo Santos-Martins, Leonardo Solis-Vasquez, et al. Gpu-accelerated drug discovery with docking on the summit supercomputer: porting, optimization, and application to covid-19 research. In *Proceedings of the 11th ACM international conference on bioinformatics, computational biology and health informatics*, pages 1–10, 2020.
- [141] Lechen Yu, Joachim Protze, Oscar Hernandez, and Vivek Sarkar. A study of memory anomalies in openmp applications. In *International Workshop on OpenMP*, pages 328–342. Springer, 2020.
- [142] Prithayan Barua, Jun Shirako, Whitney Tsang, Jeeva Paudel, Wang Chen, and Vivek Sarkar. Ompan: static verification of openmp’s data mapping constructs. In *International Workshop on OpenMP*, pages 3–18. Springer, 2019.
- [143] Lechen Yu, Joachim Protze, Oscar Hernandez, and Vivek Sarkar. Arbalest: Dynamic detection of data mapping issues in heterogeneous openmp applications. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 464–474. IEEE, 2021.

- [144] Prithayan Barua, Jisheng Zhao, and Vivek Sarkar. Ompmemopt: Optimized memory movement for heterogeneous computing. In *European Conference on Parallel Processing*, pages 200–216. Springer, 2020.
- [145] Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric, Dong H Ahn, Ignacio Laguna, Martin Schulz, Gregory L Lee, Joachim Protze, and Matthias S Müller. Archer: effectively spotting data races in large openmp applications. In *2016 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 53–62. IEEE, 2016.
- [146] SOLLVE openmp validation and verification testsuite. https://github.com/SOLLVE/sollve_vv/, 2019.
- [147] SOLLVE openmp validation and verification website. <https://crpl.cis.udel.edu/ompvvsollve/>, 2019.
- [148] Fortran Standards Committee. Information technology – Programming languages – Fortran – Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, October 2004.
- [149] Fortran Standards Committee. Information technology – Programming languages – Fortran – Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, October 2010.
- [150] Fortran Standards Committee. Draft International Standard – Information technology – Programming languages – Fortran – Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, December 2017.
- [151] LLVM Project Team. LLVM Web page. <https://llvm.org>.
- [152] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang. A Survey of Numerical Linear Algebra Methods Utilizing Mixed Precision Arithmetic, 2020. Submitted to International Journal on High Performance Computing.
- [153] Hartwig Anzt, Erik Boman, Rob Falgout, Pieter Ghysels, Michael Heroux, Xiaoye Li, Lois Curfman McInnes, Richard Tran Mills, Sivasankaran Rajamanickam, Karl Rupp, Barry Smith, Ichitaro Yamazaki, and Ulrike Meier Yang. Preparing Sparse Solvers for Exascale Computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378, January 2020.
- [154] xSDK Web page. <http://xsdk.info>.
- [155] xSDK Community Package Policies. <https://github.com/xsdk-project/xsdk-community-policies>.
- [156] GPTune. <https://github.com/gptune/GPTune>.
- [157] xSDK Code Quality. <https://github.com/xsdk-project/xsdk-code-quality>.
- [158] PETSc/TAO Team. PETSc/TAO website. <https://www.mcs.anl.gov/petsc>.
- [159] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, D. Karpeyev, D. Kaushik, M. Knepley, D. May, L. Curfman McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan, B. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Users Manual Revision 3.14. Technical Memorandum ANL-95/11 Rev. 3.14, Argonne National Laboratory, 2020.
- [160] Richard Tran Mills, Mark F Adams, Satish Balay, Jed Brown, Alp Dener, Matthew Knepley, Scott E Kruger, Hannah Morgan, Todd Munson, Karl Rupp, et al. Toward performance-portable PETSc for GPU-based exascale systems. *Parallel Computing*, page 102831, 2021.
- [161] Junchao Zhang, Jed Brown, Satish Balay, Jacob Faibussowitsch, Matthew Knepley, Oana Marin, Richard Tran Mills, Todd Munson, Barry F Smith, and Stefano Zampini. The PetscSF scalable communication layer. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

- [162] Stephen Hudson, Jeffrey Larson, John-Luke Navarro, and Stefan Wild. libEnsemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations. *IEEE Transactions on Parallel and Distributed Systems*, 2021.
- [163] Pieter Ghysels and Ryan Synk. High performance sparse multifrontal solvers on modern GPUs. *Submitted to the Special Issue A: Transitioning Libraries and Applications in Parallel Computing*, 2021.
- [164] Nan Ding, Yang Liu, Samuel Williams, and Xiaoye S. Li. *A Message-Driven, Multi-GPU Parallel Sparse Triangular Solver*, pages 147–159.
- [165] SUNDIALS Project Team. SUNDIALS Web page. <http://computation.llnl.gov/projects/sundials>.
- [166] Cody J. Balos, David J. Gardner, Carol S. Woodward, and Daniel R. Reynolds. Enabling GPU accelerated computing in the SUNDIALS time integration library. *Parallel Computing*, 108:102836, December 2021.
- [167] hypre Web page. <https://computation.llnl.gov/projects/hypre>.
- [168] R. D. Falgout, J. E. Jones, and U. M. Yang. The design and implementation of *hypre*, a library of parallel high performance preconditioners. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering*, chapter 8, pages 267–294. Springer-Verlag, 2006. UCRL-JRNL-205459.
- [169] V. Henson and U. Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.
- [170] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, September 1996. UCRL-JC-122359.
- [171] R. Li, B. Sjogreen, and U. M. Yang. A new class of amg interpolation operators based on matrix-matrix operations. 2020. Submitted to SIAM Journal on Scientific Computing.
- [172] Stanimire Tomov, Azzam Haidar, Alan Ayala, Hejer Shaiek, and Jack Dongarra. Fft-ecp implementation optimizations and features phase. ECP WBS 2.3.3.09 Milestone Report ICL-UT-19-12, FFT-ECP ST-MS-10-1440, 2019-10 2019.
- [173] Alan Ayala, Stanimire Tomov, Azzam Haidar, and Jack Dongarra. heFFTe: Highly Efficient FFT for Exascale. In *International Conference on Computational Science (ICCS 2020)*, Amsterdam, Netherlands, 2020-06 2020.
- [174] Alan Ayala, Stanimire Tomov, Miroslav Stoyanov, and Jack Dongarra. Scalability Issues in FFT Computation. In *International Conference on Parallel Computing Technologies (PaCT 2021)*. Springer, Cham, 2021-09 2021.
- [175] Alan Ayala, Stanimire Tomov, Xi Luo, Hejer Shaiek, Azzam Haidar, George Bosilca, and Jack Dongarra. Impacts of multi-gpu mpi collective communications on large fft computation. In *SC'19, Proc. of Workshop on Exascale MPI (ExaMPI)*, Denver, CO, 2019.
- [176] Hejer Shaiek, Stanimire Tomov, Alan Ayala, Azzam Haidar, and Jack Dongarra. Gpudirect mpi communications and optimizations to accelerate ffts on exascale systems. Extended Abstract icl-ut-19-06, 2019-09 2019.
- [177] Daniel Sharp, Miroslav Stoyanov, Stanimire Tomov, and Jack Dongarra. A more portable heffte: Implementing a fallback algorithm for scalable fourier transforms. ICL Tech Report ICL-UT-21-04, 2021-08 2021. HPEC'21.

- [178] Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18*, pages 47:1–47:11, Piscataway, NJ, USA, 2018. IEEE Press.
- [179] A. Sorna, X. Cheng, E. D’Azevedo, K. Won, and S. Tomov. Optimizing the fast fourier transform using mixed precision on tensor core hardware. In *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, pages 3–7, Dec 2018.
- [180] S. Tomov, J. Dongarra, and M. Baboulin. Towards dense linear algebra for hybrid gpu accelerated many-core systems. *Parallel Comput. Syst. Appl.*, 36(5-6):232–240, 2010. DOI: [10.1016/j.parco.2009.12.005](https://doi.org/10.1016/j.parco.2009.12.005).
- [181] Rajib Nath, Stanimire Tomov, and Jack Dongarra. An improved magma gemm for fermi graphics processing units. *Int. J. High Perform. Comput. Appl.*, 24(4):511–515, November 2010.
- [182] Jakub Kurzak, Stanimire Tomov, and Jack Dongarra. Autotuning GEMM kernels for the Fermi GPU. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2045–2057, November 2012.
- [183] Alan Ayala, Stanimire Tomov, Piotr Luszczek, Sebastien Cayrols, Gerald Ragghianti, and Jack Dongarra. Interim report on benchmarking fft libraries on high performance systems. ICL Tech Report ICL-UT-21-03, 2021-07 2021.
- [184] Ahmad Abdelfattah, Hartwig Anzt, Aurelien Bouteiller, Anthony Danalis, Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, Stephen Wood, Panruo Wu, Ichitaro Yamazaki, and Asim YarKhan. SLATE working note 1: Roadmap for the development of a linear algebra library for exascale computing: SLATE: Software for linear algebra targeting exascale. Technical Report ICL-UT-17-02, Innovative Computing Laboratory, University of Tennessee, June 2017. revision 04-2018.
- [185] Jakub Kurzak, Panruo Wu, Mark Gates, Ichitaro Yamazaki, Piotr Luszczek, Gerald Ragghianti, and Jack Dongarra. SLATE working note 3: Designing SLATE: Software for linear algebra targeting exascale. Technical Report ICL-UT-17-06, Innovative Computing Laboratory, University of Tennessee, September 2017. revision 09-2017.
- [186] Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. SLATE: Design of a modern distributed and accelerated linear algebra library. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC’19)*, Denver, CO, 2019. ACM.
- [187] Hartwig Anzt, Terry Cojean, Yen-Chen Chen, Goran Flegar, Fritz Göbel, Thomas Grützmacher, Pratik Nayak, Tobias Ribizel, and Yu-Hsiang Tsai. Ginkgo: A high performance numerical linear algebra library. *Journal of Open Source Software*, 5(52):2260, 2020.
- [188] Better Scientific Software (BSSw) <https://bssw.io/>.
- [189] Hartwig Anzt, Yen-Chen Chen, Terry Cojean, Jack Dongarra, Goran Flegar, Pratik Nayak, Enrique S. Quintana-Ortí, Yuhsiang M. Tsai, and Weichung Wang. Towards continuous benchmarking: An automated performance evaluation framework for high performance software. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC ’19*, pages 9:1–9:11, New York, NY, USA, 2019. ACM.
- [190] Yuhsiang M. Tsai, Terry Cojean, and Hartwig Anzt. Porting a sparse linear algebra math library to intel gpus, 2021.
- [191] Fritz Göbel, Thomas Grützmacher, Tobias Ribizel, and Hartwig Anzt. Mixed precision incomplete and factorized sparse approximate inverse preconditioning on gpus. In Leonel Sousa, Nuno Roma, and Pedro Tomás, editors, *Euro-Par 2021: Parallel Processing*, pages 550–564, Cham, 2021. Springer International Publishing.

- [192] José Ignacio Aliaga, Hartwig Anzt, Thomas Grützmacher, Enrique S. Quintana-Ortí, and Andrés E. Tomás. Compressed basis GMRES on high performance gpus. *CoRR*, abs/2009.12101, 2020.
- [193] Isha Aggarwal, Aditya Kashi, Pratik Nayak, Cody J. Balos, Carol S. Woodward, and Hartwig Anzt. Batched sparse iterative solvers for computational chemistry simulations on gpus. In *2021 12th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, pages 35–43, 2021.
- [194] William F. Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, Axel Huebl, Mark Kim, James Kress, Tahsin Kurc, Qing Liu, Jeremy Logan, Kshitij Mehta, George Ostrouchov, Manish Parashar, Franz Poeschel, David Pugmire, Eric Suchyta, Keichi Takahashi, Nick Thompson, Seiji Tsutsumi, Lipeng Wan, Matthew Wolf, Kesheng Wu, and Scott Klasky. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.
- [195] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, et al. Hello ADIOS: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience*, 26(7):1453–1473, 2014.
- [196] ADIOS2 documentation. <https://adios2.readthedocs.io>.
- [197] The ADIOS2 framework. <https://github.com/ornladios/ADIOS2>.
- [198] James Ahrens, Kristi Brislawn, Ken Martin, Berk Geveci, C. Charles Law, and Michael Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications*, 21(4):34–41, July/August 2001.
- [199] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, May/June 2010. DOI 10.1109/MCG.2010.51.
- [200] Kenneth Moreland. The ParaView tutorial, version 4.4. Technical Report SAND2015-7813 TR, Sandia National Laboratories, 2015.
- [201] Kenneth Moreland. Oh, \$#*%! Exascale! The effect of emerging architectures on scientific discovery. In *2012 SC Companion (Proceedings of the Ultrascale Visualization Workshop)*, pages 224–231, November 2012. DOI 10.1109/SC.Companion.2012.38.
- [202] Kenneth Moreland, Berk Geveci, Kwan-Liu Ma, and Robert Maynard. A classification of scientific visualization algorithms for massive threading. In *Proceedings of Ultrascale Visualization Workshop*, November 2013.
- [203] Kenneth Moreland, Christopher Sewell, William Usher, Li ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, Matthew Larsen, Chun-Ming Chen, Robert Maynard, and Berk Geveci. VTK-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May/June 2016. DOI 10.1109/MCG.2016.48.
- [204] Kenneth Moreland. The vtk-m user’s guide. techreport SAND 2018-0475 B, Sandia National Laboratories, 2018. <http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf>.
- [205] Guy E. Blelloch. *Vector Models for Data-Parallel Computing*. MIT Press, 1990. ISBN 0-262-02313-X.
- [206] Li-ta Lo, Chris Sewell, and James Ahrens. PISTON: A portable cross-platform framework for data-parallel visualization operators. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2012. DOI 10.2312/EGPGV/EGPGV12/011-020.
- [207] Matthew Larsen, Jeremy S. Meredith, Paul A. Navrátil, and Hank Childs. Ray tracing within a data parallel framework. In *IEEE Pacific Visualization Symposium (PacificVis)*, April 2015. DOI 10.1109/PACIFICVIS.2015.7156388.

- [208] Kenneth Moreland, Matthew Larsen, and Hank Childs. Visualization for exascale: Portable performance is critical. In *Supercomputing Frontiers and Innovations*, volume 2, 2015. DOI 10.2312/pgv.20141083.
- [209] Kenneth Moreland, Robert Maynard, David Pugmire, Abhishek Yenpure, Allison Vacanti, Matthew Larsen, and Hank Childs. Minimizing development costs for efficient many-core visualization using MCD³. *Parallel Computing*, 108(102834), December 2021.
- [210] H. Carter Edwards, Daniel Sunderland, Chris Amsler, and Sam Mish. Multicore/GPGPU portable computational kernels via multidimensional arrays. In *IEEE Cluster*, September 2011.
- [211] Jeremy S. Meredith, Sean Ahern, Dave Pugmire, and Robert Sisneros. EAVL: The extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 21–30, 2012. DOI 10.2312/EGPGV/EGPGV12/021-030.
- [212] Kenneth Moreland, Brad King, Robert Maynard, and Kwan-Liu Ma. Flexible analysis software for emerging architectures. In *2012 SC Companion (Petascale Data Analytics: Challenges and Opportunities)*, pages 821–826, November 2012. DOI 10.1109/SC.Companion.2012.115.
- [213] Sudhanshu Sane, Abhishek Yenpure, Roxana Bujack, Matthew Larsen, Kenneth Moreland, Christoph Garth, Chris R. Johnson, and Hank Childs. Scalable in situ computation of lagrangian representations via local flow maps. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, June 2021.
- [214] Sudhanshu Sane, Chris R. Johnson, and Hank Childs. Investigating in situ reduction via lagrangian representations for cosmology and seismology applications. In *Computational Science – ICCS 2021*, pages 436–450, June 2021.
- [215] Utkarsh Ayachit. The ParaView guide: a parallel visualization application, 2015.
- [216] James Ahrens, Berk Geveci, and Charles Law. ParaView: An end-user tool for large-data visualization. *The visualization handbook*, 2005.
- [217] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 357–372. CRC Press/Francis–Taylor Group, October 2012.
- [218] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the Third Workshop of In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), held in conjunction with SC17*, Denver, CO, USA, October 2017.
- [219] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O’Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV 2015)*, pages 25–29, November 2015.
- [220] Hamish A. Carr, Gunther H. Weber, Christopher M. Sewell, Oliver Rübel, Patricia Fasel, and James P. Ahrens. Scalable contour tree computation by data parallel peak pruning. *Transactions on Visualization and Computer Graphics*, 2019. In press.
- [221] A Biswas, S Dutta, J Pulido, and J Ahrens. In situ data-driven adaptive sampling for large-scale simulation data summarization. In *In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV*, pages 13–18, 2018.
- [222] Soumya Dutta, Ayan Biswas, and James Ahrens. Multivariate pointwise information-driven data sampling and visualization. *Entropy*, 21(7):1–25, 2019.

- [223] Qun Liu, Subhashis Hazarika, John M. Patchett, James P. Ahrens, and Ayan Biswas. Poster: Deep learning-based feature-aware data modeling for complex physics simulations. In *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 11 2019. To Appear in SC '19.
- [224] S. Dutta, J. Woodring, Han-Wei Shen, J. Chen, and J. Ahrens. Homogeneity guided probabilistic data summaries for analysis and visualization of large-scale data sets. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pages 111–120, 2017.
- [225] Soumya Dutta and Han-Wei Shen. Distribution driven extraction and tracking of features for time-varying data analysis. *IEEE Trans. on Vis. and Comp. Graphics*, 22(1):837–846, 2016.
- [226] Aaditya G Landge, Valerio Pascucci, Attila Gyulassy, Janine C Bennett, Hemanth Kolla, Jacqueline Chen, and Peer-Timo Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031. IEEE, 2014.
- [227] Steve Petruzza, Sean Treichler, Valerio Pascucci, and Peer-Timo Bremer. Babelflow: An embedded domain specific language for parallel analysis and visualization. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 463–473. IEEE, 2018.
- [228] Xavier Bonaventura, Miquel Feixas, Mateu Sbert, Lewis Chuang, and Christian Wallraven. A survey of viewpoint selection methods for polygonal models. *Entropy*, 20(5), 2018.
- [229] Nicole Marsaglia. Automatic camera selection for in situ visualization. Technical Report AREA-202001-Marsaglia, University of Oregon, Computer and Information Sciences Department, 1 2020. Available at <https://www.cs.uoregon.edu/Reports/AREA-202001-Marsaglia.pdf>.
- [230] Sudhanshu Sane, Roxana Bujack, and Hank Childs. Revisiting the Evaluation of In Situ Lagrangian Analysis. In Hank Childs and Fernando Cucchietti, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2018.
- [231] Sudhanshu Sane, Hank Childs, and Roxana Bujack. An Interpolation Scheme for VDVP Lagrangian Basis Flows. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 109–118, Porto, Portugal, June 2019.
- [232] Roba Binyahib, David Pugmire, Boyana Norris, and Hank Childs. A Lifeline-Based Approach for Work Requesting and Parallel Particle Advection. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, Vancouver, Canada, October 2019.
- [233] Duong Hoang, Harsh Bhatia, Peter Lindstrom, and Valerio Pascucci. High-quality and low-memory-footprint progressive decoding of large-scale particle data. In *IEEE Symposium on Large Data Analysis and Visualization*, 2021. To appear.
- [234] Stephen Hamilton, Randal Burns, Charles Meneveau, Perry Johnson, Peter Lindstrom, John Patchett, and Alexander Szalay. Extreme event analysis in next generation simulation architectures. In *ISC High Performance 2017*, pages 277–293, 2017.
- [235] Peter Lindstrom. Error distributions of lossy floating-point compressors. In *JSM 2017 Proceedings*, pages 2574–2589, 2017.
- [236] James Diffenderfer, Alyson Fox, Jeffrey Hittinger, Geoffrey Sanders, and Peter Lindstrom. Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing*, 41(3):A1867–A1898, 2019.
- [237] Dorit Hammerling, Allison Baker, Alexander Pinard, and Peter Lindstrom. A collaborative effort to improve lossy compression methods for climate data. In *5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5)*, Nov 2019.

- [238] Lennart Noordsij, Steven van der Vlugt, Mohamed Bamakhrama, Zaid Al-Ars, and Peter Lindstrom. Parallelization of variable rate decompression through metadata. In *Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 245–252, Mar 2020.
- [239] Duong Hoang, Brian Summa, Harsh Bhatia, Peter Lindstrom, Pavol Klacansky, Will Usher, Peer-Timo Bremer, and Valerio Pascucci. Efficient and flexible hierarchical data layouts for a unified encoding of scalar field precision and resolution. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):603–613, 2021.
- [240] Franck Cappello and Peter Lindstrom. Compression of scientific data. ISC High Performance 2017 Tutorials, 2017.
- [241] Franck Cappello and Peter Lindstrom. Compression of scientific data. IEEE/ACM SC 2017 Tutorials, 2017.
- [242] Franck Cappello and Peter Lindstrom. Compression for scientific data. Euro-Par 2018 Tutorials, 2018.
- [243] Franck Cappello and Peter Lindstrom. Compression for scientific data. IEEE/ACM SC 2018 Tutorials, 2018.
- [244] Franck Cappello and Peter Lindstrom. Compression for scientific data. ISC High Performance 2019 Tutorials, 2019.
- [245] Franck Cappello, Peter Lindstrom, and Sheng Di. Compression for scientific data. IEEE/ACM SC 2019 Tutorials, 2019.
- [246] Franck Cappello, Peter Lindstrom, and Sheng Di. Lossy compression for scientific data. IEEE/ACM SC 2020 Tutorials, 2020.
- [247] Franck Cappello, Peter Lindstrom, Sheng Di, Jon Calhoun, Pascal Grosset, Katrin Heitmann, and Allison Baker. Lossy compression for scientific data: Success stories. IEEE/ACM SC 2020 Panels, 2020.
- [248] Franck Cappello, Peter Lindstrom, and Sheng Di. Lossy compression for scientific data. IEEE/ACM SC 2021 Tutorials, 2021.
- [249] E4S Web page. <http://e4s.io>.
- [250] E4S Validation Test Suite. <https://github.com/E4S-Project/testsuite>.
- [251] ExaWorks SDK Community Policies. <https://github.com/ExaWorks/SDK/blob/master/POLICIES.md>.
- [252] Job API Specification Github Project. <https://github.com/ExaWorks/job-api-spec>.
- [253] J/PSI Python Implementation Github Project. <https://github.com/ExaWorks/psi-j-python>.
- [254] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Ilkay Altintas, Rosa M Badia, Bartosz Balis, Tainã Coleman, Frederik Coppens, Frank Di Natale, Bjoern Enders, Thomas Fahringer, Rosa Filgueira, Grigori Fursin, Daniel Garijo, Carole Goble, Dorran Howell, Shantenu Jha, Daniel S. Katz, Daniel Laney, Ulf Leser, Maciej Malawski, Kshitij Mehta, Loïc Pottier, Jonathan Ozik, J. Luc Peterson, Lavanya Ramakrishnan, Stian Soiland-Reyes, Douglas Thain, and Matthew Wolf. A community roadmap for scientific workflows research and development, 2021.
- [255] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Dan Laney, Dong Ahn, Shantenu Jha, Carole Goble, Lavanya Ramakrishnan, Luc Peterson, Bjoern Enders, Douglas Thain, Ilkay Altintas, Yadu Babuji, Rosa Badia, Vivien Bonazzi, Taina Coleman, Michael Crusoe, Ewa Deelman, Frank Di Natale, Paolo Di Tommaso, Thomas Fahringer, Rosa Filgueira, Grigori Fursin, Alex Ganose, Bjorn Gruning, Daniel S. Katz, Olga Kuchar, Ana Kupresanin, Bertram Ludascher, Ketan Maheshwari, Marta Mattoso, Kshitij Mehta, Todd Munson, Jonathan Ozik, Tom Peterka, Loic Pottier, Tim Randles, Stian Soiland-Reyes, Benjamin Tovar, Matteo Turilli, Thomas Uram, Karan Vahi, Michael Wilde, Matthew Wolf, and Justin Wozniak. Workflows Community Summit: Bringing the Scientific Workflows Community Together, March 2021.

- [256] Rafael Ferreira da Silva, Henri Casanova, Kyle Chard, Tainã Coleman, Dan Laney, Dong Ahn, Shantenu Jha, Dorran Howell, Stian Soiland-Reys, Ilkay Altintas, Douglas Thain, Rosa Filgueira, Yadu Babuji, Rosa M. Badia, Bartosz Balis, Silvina Caino-Lores, Scott Callaghan, Frederik Coppens, Michael R. Crusoe, Kaushik De, Frank Di Natale, Tu M. A. Do, Bjoern Enders, Thomas Fahringer, Anne Fouilloux, Grigori Fursin, Alban Gaignard, Alex Ganose, Daniel Garijo, Sandra Gesing, Carole Goble, Adil Hasan, Sebastiaan Huber, Daniel S. Katz, Ulf Leser, Douglas Lowe, Bertram Ludaescher, Ketan Maheshwari, Maciej Malawski, Rajiv Mayani, Kshitij Mehta, Andre Merzky, Todd Munson, Jonathan Ozik, Loïc Pottier, Sashko Ristov, Mehdi Roozmeh, Renan Souza, Frédéric Suter, Benjamin Tovar, Matteo Turilli, Karan Vahi, Alvaro Vidal-Torreira, Wendy Whitcup, Michael Wilde, Alan Williams, Matthew Wolf, and Justin Wozniak. Workflows Community Summit: Advancing the State-of-the-art of Scientific Workflows Management Systems Research and Development, June 2021.
- [257] Aymen Al-Saadi, Dong H. Ahn, Yadu Babuji, Kyle Chard, James Corbett, Mihael Hategan, Stephen Herbein, Shantenu Jha, Daniel Laney, Andre Merzky, Todd Munson, Michael Salim, Mikhail Titov, Matteo Turilli, and Justin M. Wozniak. Exaworks: Workflows for exascale, 2021.
- [258] E4S Developers. E4S Build Cache for Spack. <https://oaciss.uoregon.edu/e4s/inventory.html/>.
- [259] R Priedhorsky and TC Randles. Charliecloud: Unprivileged containers for user-defined software stacks in hpc. Technical Report LA-UR-16-22370, Los Alamos National Laboratory, 2016. available as <http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-16-22370>.
- [260] Docker Inc. Docker. <https://www.docker.com>.
- [261] BEE. BEE. <http://bee.dsscale.org>.
- [262] RS Canon and D Jacobsen. Shifter: Containers for hpc. In *Proceedings of the Cray User's Group*, 2016.
- [263] Bauer MW Kurtzer GM, Sochat V. Singularity: Scientific containers for mobility of compute. *PLoS ONE*, may 2017. available as <https://doi.org/10.1371/journal.pone.0177459>.
- [264] Tao B. Schardl, William S. Moses, and Charles E. Leiserson. Tapir: Embedding fork-join parallelism into llvm's intermediate representation. *SIGPLAN Not.*, 52(8):249–265, January 2017.
- [265] J. Ahrens, S. Jourdain, P. OLeary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434, Nov 2014.
- [266] S. Garcia and M. Sukop and K. Cunningham. Groundwater Dataset. https://www.researchgate.net/figure/a-d-Limestone-samples-and-macropore-types-for-which-high-resolution-X-ray-CT-scans-were_fig4_258816440. The dataset contains a limestone ground sample from south Florida, and streamlines that represent the flow of water through the cavities. Data courtesy of Sadé Garcia, Michael Sukop, Florida International University; Kevin Cunningham, United States Geological Survey.
- [267] Pantheon Developers. Pantheon Project Website. <http://pantheonscience.org/>.
- [268] M A Sprague, S Ananthan, G Vijayakumar, and M Robinson. ExaWind: A multifidelity modeling and simulation environment for wind energy. 1452:012071, jan 2020.
- [269] Ethan Stam. Pantheon/e4s/exawind-naluwind. Citation is for Pantheon workflow only.
- [270] Adam J. Stewart, Massimiliano Culpo, Gregory Becker, Peter Scheibel, and Todd Gamblin. Spack Community BoF. In *Supercomputing 2019*, Denver, CO, November 21 2019.
- [271] Todd Gamblin, Gregory Becker, Massimiliano Culpo, Mario Melara, Peter Scheibel, and Adam J. Stewart. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2019*, Denver, CO, November 18 2019. Full day.

- [272] Todd Gamblin and Gregory Becker. Tutorial: Spack for Developers. In *Los Alamos National Laboratory*, Los Alamos, NM, November 5 2019. Full day.
- [273] Todd Gamblin and Gregory Becker. Spack Tutorial. In *1st Workshop on NSF and DOE High Performance Computing Tools*, Eugene, OR, July 10-11 2019. University of Oregon.
- [274] Levi Baber, Adam J. Stewart, Gregory Becker, and Todd Gamblin. Tutorial: Managing HPC Software Complexity with Spack. In *Practice and Experience in Advanced Research Computing (PEARC'19)*, Chicago, IL, July 31 2019. Half day.
- [275] Todd Gamblin, Gregory Becker, Massimiliano Culpo, and Michael Kühn. Tutorial: Managing HPC Software Complexity with Spack. In *ISC High Performance*, Houston, TX, June 16 2019. Half day.
- [276] Todd Gamblin, Gregory Becker, Matthew P. LeGendre, and Peter Scheibel. Spack Roundtable Discussion. In *Exascale Computing Project 3rd Annual Meeting*, Houston, TX, January 16 2019.
- [277] Todd Gamblin, Gregory Becker, Peter Scheibel, Matt Legendre, and Mario Melara. Managing HPC Software Complexity with Spack. In *Exascale Computing Project 3rd Annual Meeting*, Houston, TX, January 14 2019. Full day.
- [278] Todd Gamblin, Adam Stewart, Johannes Albert von der Gönna an Marc Pérache, and Matt Belhorn. Spack Community Birds-of-a-Feather Session. In *Supercomputing 2018*, Dallas, TX, November 13 2018.
- [279] Todd Gamblin, Gregory Becker, Massimiliano Culpo, Gregory L. Lee, Matt Legendre, Mario Melara, and Adam J. Stewart. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2018*, Dallas, TX, November 12 2018. Full day.
- [280] Todd Gamblin, William Scullin, Matt Belhorn, Mario Melara, and Gerald Ragghianti. Spack State of the Union. In *Exascale Computing Project 2nd Annual Meeting*, Knoxville, TN, February 6-8 2018.
- [281] Todd Gamblin, Gregory Becker, Massimiliano Culpo, Gregory L. Lee, Matt Legendre, Mario Melara, and Adam J. Stewart. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2017*, Salt Lake City, Utah, November 13 2017. Full day.
- [282] Todd Gamblin. Tutorial: Managing HPC Software Complexity with Spack. In *HPC Knowledge Meeting (HPCKP'17)*, San Sebastián, Spain, June 16 2017. 2 hours.
- [283] Gregory Becker, Matt Legendre, and Todd Gamblin. *Tutorial: Spack for HPC*. Livermore Computing, Lawrence Livermore National Laboratory, Livermore, CA, April 6 2017. Half day.
- [284] Todd Gamblin, Massimiliano Culpo, Gregory Becker, Matt Legendre, Greg Lee, Elizabeth Fischer, and Benedikt Hegner. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2016*, Salt Lake City, Utah, November 13 2016. Half day.
- [285] MFEM: Modular finite element methods library. mfem.org.
- [286] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny V. Dobrev, Y. Dudouit, A. Fisher, Tz. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini. MFEM: A modular finite element library. *Computers & Mathematics with Applications*, 2020.
- [287] BLAST: High-order finite element Lagrangian hydrocode. <https://computation.llnl.gov/projects/blast>.
- [288] Flux Framework Community. Flux Framework: A flexible framework for resource management customized for your HPC site. <http://flux-framework.org>, October 2021. (Retrieved Oct. 1, 2021).
- [289] Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Helgi I. Ingólfsson, Joseph Koning, Tapasya Patki, Thomas R.W. Scogland, Becky Springmeyer, and Michela Taufer. Flux: Overcoming scheduling challenges for exascale workflows. *Future Generation Computer Systems*, 110:202–213, 2020.

- [290] R. W. Anderson, V. A. Dobrev, T. V. Kolev, R. N. Rieben, and V. Z. Tomov. High-order multi-material ALE hydrodynamics. *SIAM J. Sc. Comp.*, 40(1):B32–B58, 2018.
- [291] V. A. Dobrev, T. V. Kolev, D. Kuzmin, R. N. Rieben, and V. Z. Tomov. Sequential limiting in continuous and discontinuous Galerkin methods for the Euler equations. *J. Comput. Phys.*, 356:372–390, 2018.
- [292] R. W. Anderson, V. A. Dobrev, T. V. Kolev, D. Kuzmin, M. Quezada de Luna, R. N. Rieben, and V. Z. Tomov. High-order local maximum principle preserving (MPP) discontinuous Galerkin finite element method for the transport equation. *J. Comput. Phys.*, 334:102–124, 2017.
- [293] V. A. Dobrev, T. V. Kolev, R. N. Rieben, and V. Z. Tomov. Multi-material closure model for high-order finite element Lagrangian hydrodynamics. *Int. J. Numer. Meth. Fluids*, 82(10):689–706, 2016.
- [294] V. A. Dobrev, T. V. Kolev, and R. N. Rieben. High order curvilinear finite elements for elastic-plastic Lagrangian dynamics. *J. Comput. Phys.*, 257, Part B:1062 – 1080, 2014.
- [295] V. A. Dobrev, T. E. Ellis, Tz. V. Kolev, and R. N. Rieben. High-order curvilinear finite elements for axisymmetric Lagrangian hydrodynamics. *Computers and Fluids*, 83:58–69, 2013.
- [296] V. A. Dobrev, T. V. Kolev, and R. N. Rieben. High-order curvilinear finite element methods for Lagrangian hydrodynamics. *SIAM J. Sc. Comp.*, 34(5):B606–B641, 2012.
- [297] V. A. Dobrev, T. E. Ellis, Tz. V. Kolev, and R. N. Rieben. Curvilinear finite elements for Lagrangian hydrodynamics. *Int. J. Numer. Meth. Fluids*, 65(11-12):1295–1310, 2011.
- [298] James Ahrens, Berk Geveci, and Charles Law. ParaView: An end-user tool for large data visualization. In *Visualization Handbook*. Elsevier, 2005. ISBN 978-0123875822.