EXASCALE COMPUTING PROJECT

ECP-RPT-ST-0002-2020–Public

ECP Software Technology Capability Assessment Report–Public

Michael A. Heroux, Director ECP ST
Jonathan Carter, Deputy Director ECP ST
Rajeev Thakur, Programming Models & Runtimes Lead
Jeffrey S. Vetter, Development Tools Lead
Lois Curfman McInnes, Mathematical Libraries Lead
James Ahrens, Data & Visualization Lead
Todd Munson, Software Ecosystem & Delivery Lead
J. Robert Neely, NNSA ST Lead

February 1, 2020

U.S. DEPARTMENT OF ENERGY | Office of Science

National Nuclear Security Administration

ECP-RPT-ST-0002-2020–Public

ECP Software Technology Capability Assessment Report–Public

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

February 1, 2020

REVISION LOG

Version	Date	Description
1.0	July 1, 2018	*ECP ST Capability Assessment Report*
1.5	February 1, 2019	*Second release*
2.0	February 1, 2020	*Third release*

EXECUTIVE SUMMARY

The Exascale Computing Project (ECP) Software Technology (ST) Focus Area is responsible for developing critical software capabilities that will enable successful execution of ECP applications, and for 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 (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 six to 12 months). Highlights from this version of the report are presented here.

What is new in CAR V2.0: CAR V2.0 contains the following updates relative to CAR V1.5.

- We introduce the FY20–23 project structure. ECP ST now consists of 6 (up from 5) level-3 (L3) technical areas, introducing the NNSA ST L3 area, which brings into one L3 the ECP open source development efforts at NNSA labs that are of particular importance to the rest of ECP. The number of ECP ST level-4 (L4) subprojects has been reduced from 55 to 33. The strategic aggregation of projects into fewer and larger units enables us to better manage L4 subprojects consistently as a portfolio. See Section 1.2.

- We describe new and enhanced project management processes and resources including our iterative planning process, new KPP-3 capability integration process, a product dictionary and dependency management database. These new project features and related dashboards enable more insight and better information to effectively manage efforts across ECP. See Section 2.

- The two-page summaries of each ECP L4 projects have been updated to reflect recent progress and next steps. See Section 4.

- The Extreme-scale Scientific Software Stack (E4S) is further described. The third release, which is also the first major public release Version 1.0, was November 18, 2019. E4S is the primary integration and delivery vehicle for ECP ST capabilities. See Section 2.1.1.

- The ECP ST SDK effort has further refined its groupings. See Section 2.1.2.

The Exascale Computing Project Software Technology (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 70 software products (Section 2.1.3) in six technical areas (Table 1). Since the publishing of CAR V1.5, we have introduced a 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, we are 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 the need to provide vendor-specific implementations for each exascale system. We anticipate that one legacy of ECP ST efforts will be a software stack that supports Intel and AMD accelerators in addition to Nvidia. See Section 4.1.

Development Tools: We are enhancing existing widely used compilers (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 even more necessary, compilers must generate optimized code for many architectures, and the impediments to performance and scalability become even harder 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, utilize new memory systems and more. See Section 4.2.

Mathematical Libraries: High-performance scalable math libraries have enabled parallel execution of many applications for decades. ECP ST is providing the next generation of these libraries to address needs for latency hiding, improved vectorization, threading and strong scaling. In addition, we are addressing

new demands for system-wide scalability including improved support for coupled systems and ensemble calculations. See Section 4.3. The math libraries teams are also spearheading the software development kit (SDK) initiative that is a pillar of the ECP ST software delivery strategy (Section 2.1.2).

Data & Visualization: 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 we produce grows faster than our ability to capture and interpret it. See Section 4.4.

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

NNSA ST: This technical area brings into one L3 area all of the NNSA-funded work in ECP ST for easier coordination with other project work at the NNSA labs. Introducing this L3 enables continued integrated planning with the rest of ECP ST while permitting flexible coordination within the NNSA labs. See Section 4.6.

ECP ST Software Delivery mechanisms: 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. Each of the major DOE computing facilities provides direct support of some users for about 20 ECP ST products. About 10 products are available via vendor software stack and via binary distributions such as Linux distributions.

ECP ST Project Restructuring: ECP ST completed a significant restructuring in October 2019 (Section 1.2). We increased the number of technical areas from 5 to 6 and reduced the number of L4 projects from 55 to 33. We introduced a new L4 subproject for software packaging and delivery in the L3 technical area (SW Ecosystem & Delivery) that enhances existing NNSA funding for Spack and creates specific funding and goals for container environments.

ECP ST Project Overviews: A significant portion of this report includes 2-page synopses of each ECP ST project (Section 4), including a project overview, key challenges, solution strategy, recent progress and next steps.

Project organization: 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 33 projects and more than 250 contributors to create the ECP ST software stack.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The Exascale Computing Project Software Technology (ECP ST) focus area represents the key bridge between Exascale systems and the scientists developing applications that will run on those platforms. 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. But 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 approaching scientific computing in a holistic manner, encompassing the entire user workflow—from conception of a problem, 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 conceive of new approaches necessary to address unique problems where 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 for the research, development, and delivery of a comprehensive software stack, as summarized in Table 1.

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 spanning 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 bottom up based on application requirements and the existing software stack at DOE HPC Facilities. The portfolio comprises projects selected in two different ways:

1. Thirty projects funded by the DOE Office of Science (ASCR). This scope of work was selected in October 2016 via an RFI and RFP process, considering prioritized requirements. The initial collection of loosely coupled projects has been re-organized twice and is now in a form that should serve us well as we move to the more formal execution phases of the project.

2. Three DOE NNSA/ASC funded projects that are part of the Advanced Technology Development and Mitigation (ATDM) program, which is in its sixth year (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/ASC simulation codes.

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

1.1 BACKGROUND

Historically, the software used on supercomputers has come from three sources: computer system vendors, DOE laboratories, and academia. Traditionally, vendors have supplied system software: operating system, 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 to improve management of the systems. An observation is that it is common for system software to break or 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 DOE laboratories and universities and ported to the new computer architectures when they are deployed. These math 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 that 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. An observation is that the vendor is ultimately responsible for providing a programming environment and file system with the supercomputer, but there is often a struggle to get the vendors to support software developed by others or to invest in new ideas that have few or no users yet. Another observation is that file-system software plays a key role in overall

WBS 2.3.1	Programming Models and Runtimes	Cross-platform, production-ready programming infrastructure to support development and scaling of mission-critical software at both 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 [5] as a delivery and deployment vehicle.
WBS 2.3.3	Mathematical Libraries	Mathematical libraries and frameworks that (i) interoperate with the ECP software stack; (ii) are incorporated into ECP applications; and (iii) provide scalable, resilient numerical algorithms that facilitate efficient simulations on Exascale computers.
WBS 2.3.4	Data and Visualization	Production infrastructure necessary to manage, share, and facilitate analysis and visualization of data in support of mission-critical codes. Data analytics and visualization software that supports 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), the Extreme-scale Scientific Software Stack (E4S) across all of ECP ST projects. Development of capabilities in Spack [1] in collaboration with NNSA's primary sponsorship. Development of SuperContainers [6] 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.

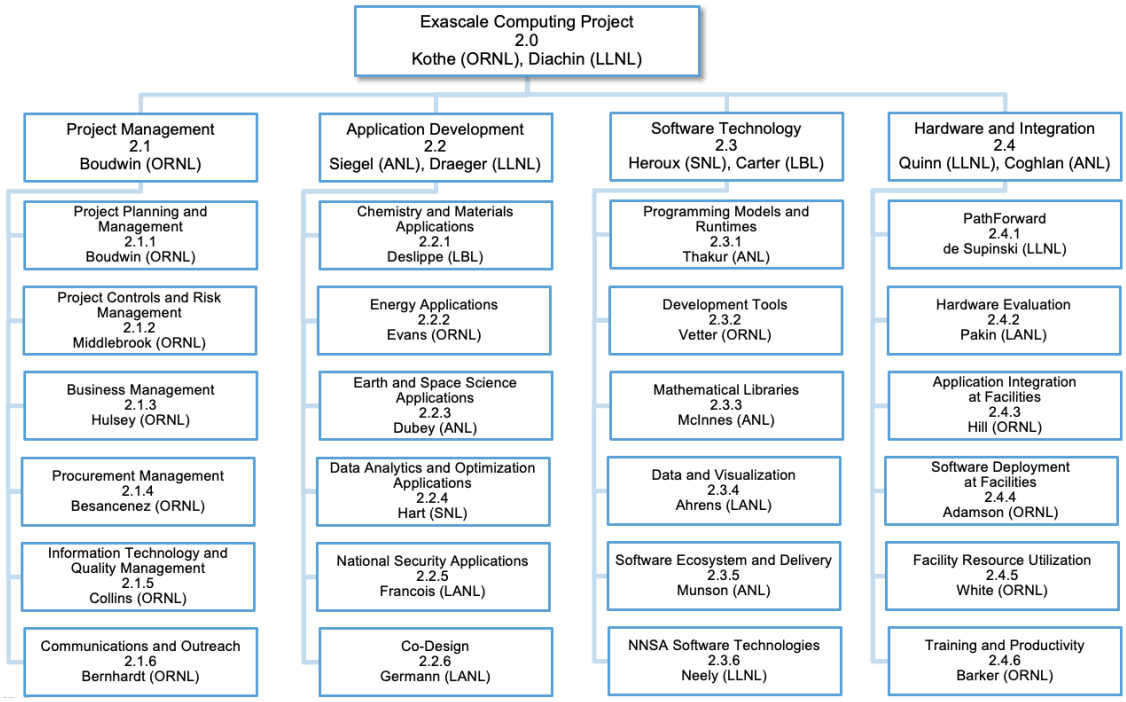


Figure 1: The ECP Work Breakdown Structure through Level 3 (L3) as of December 5, 2019. Under Software Technology, WBS 2.3.6 consolidates ATDM contributions to ECP into a new L3 area.

system resilience, and the difficulty of making the file-system software resilient has grown non-linearly 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. Because clock speeds have essentially stalled, the 1000-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 coordinated transfer of data and work between the host and device. While some of this transfer can be managed implicitly, for the most performance-sensitive phases, the programmer typically must manage host-device coordination explicitly. Much of the software transformation effort will be focused on this issue.

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

- **2021 acceleration:** ECP has a goal of accelerating the development of the U.S. Exascale systems and enabling the first deployment by 2021. This means that the software needs to be ready sooner, and the approach of just waiting until it is ready will not work. A concerted plan that accelerates the development of the highest priority and most impactful software is needed.

- **Productivity:** Traditional supercomputer software requires a great deal of expertise to use. ECP has a goal of making Exascale computing accessible to a wider science community than previous supercomputers have been. This requires the development of 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 will need to 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 employed by ECP ST to address the Exascale challenges.

1.2 ECP ST PROJECT RESTRUCTURING

The initial organization of ECP ST was based on discussions that occurred over several years of Exascale planning within DOE, especially the DOE Office of Advanced Scientific Computing Research (ASCR). Figure 4 shows the conceptual diagram of this first phase. The 66 ECP ST projects were mapped into 8 technical areas, in some cases arbitrating where a project should go based on its primary type of work, even if other work was present in the project. In November 2017, ECP ST was reorganized into 5 technical areas, primarily through merging a few smaller areas, and the number of projects was reduced to 56 (then 55 due to further merging in SW Ecosystem & Delivery). Figure 5 shows the diagram of the second phase of ECP ST. In Section 2, we describe the organization, planning, execution, tracking and assessment processes that will put ECP ST in a good position for success in the CD-2 phase of the project.

2. ECP SOFTWARE TECHNOLOGY PLANNING, EXECUTION, TRACKING AND ASSESSMENT

During the past two years, ECP ST has introduced the Extreme-scale Scientific Software Stack (E4S) and Software Development Kits (SDKs). We have established new approaches for project planning, execution, tracking and assessment using a tailored earned value management system that enables iterative and incremental refinement to its planning process. We have also revised our key performance parameter (KPP-3, the third of ECP's four KPPs) to be solely focused on measuring capability integration into client environments. We have developed and used an assessment process that has led to significant changes in the number and scope of L4 subprojects.

2.1 ECP SOFTWARE TECHNOLOGY ARCHITECTURE AND DESIGN

ECP is taking an approach of codesign across all its principal technical areas: applications development (AD), software technology (ST), and hardware & integration (HI). For ECP ST, this means its requirements are based on input from other areas, and there is a tight integration of the software products both within the software stack as well as with applications and the evolving hardware.

The portfolio of projects in ECP ST is intended to address the Exascale challenges and requirements described above. We note that ECP is not developing the entire software stack for an Exascale system. For example, we expect vendors 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 system, file system, compilers for C, C++, Fortran, etc. (increasingly derived from the LLVM community ecosystem to which ECP contributes), basic math libraries, system monitoring tools, scheduler, debuggers, vendor's performance tools, MPI (based on ECP-funded projects), OpenMP (with features from ECP-funded project), and data-centric stack components. ECP develops other, mostly higher-level software that is needed

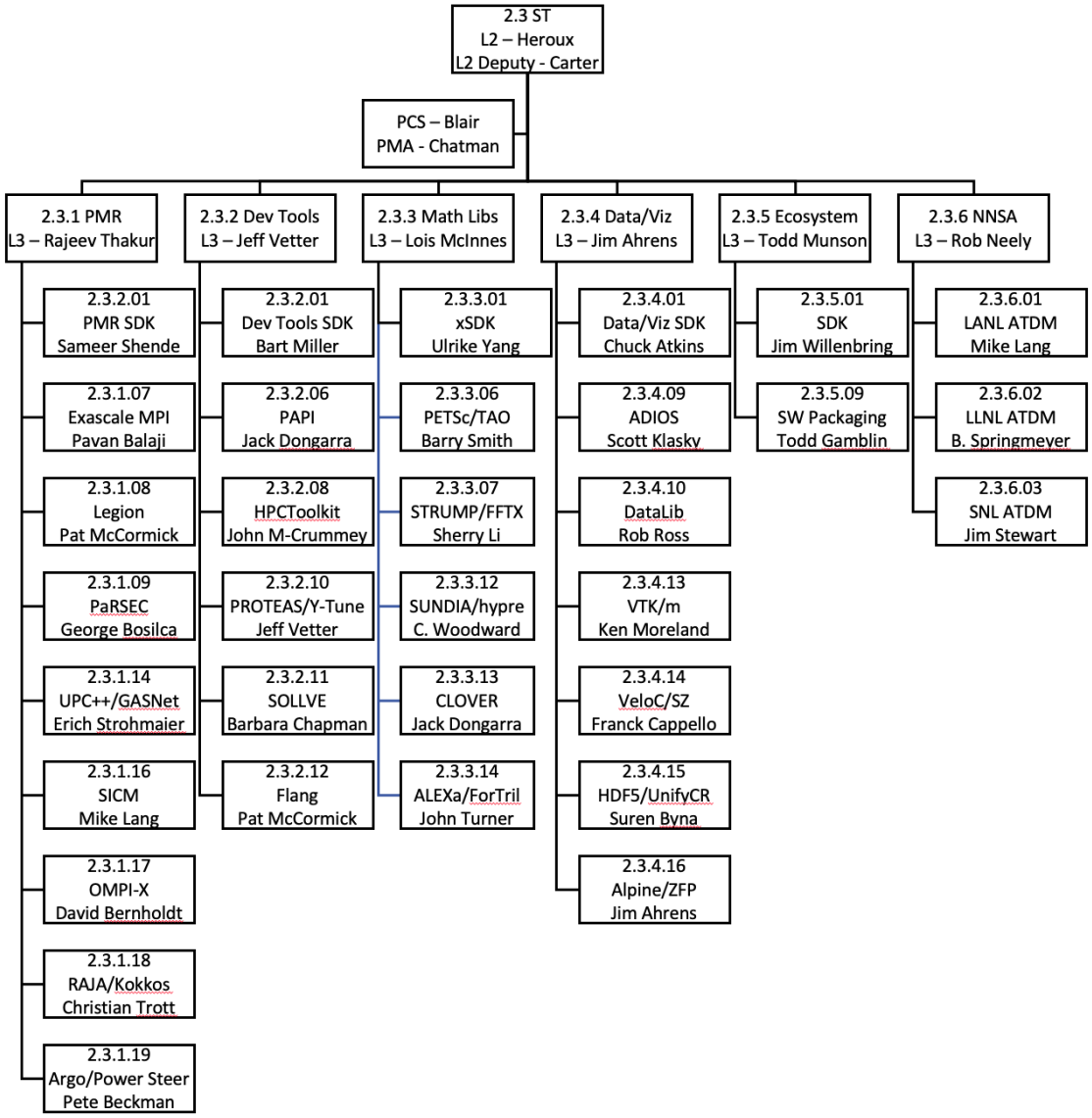


Figure 2: The FY20 ECP ST WBS structure as of December 5, 2019, includes a new L3 (2.3.6) and better balances L4 subprojects in the first four L3 technical areas. Technical area 2.3.5 has only two projects, which are focused on meta-product development in SDKs, E4S, Spack and SuperContainers.

- Phase 1: 66 total L4 subprojects
  - 35 projects funded by the DOE Office of Science that were selected in late 2016 via an RFI and RFP process, considering prioritized requirements of applications and DOE facilities. These projects started work in January–March 2017 depending on when the contracts were awarded.
  - 31 ongoing DOE NNSA funded projects that are part of the Advanced Technology Development and Mitigation (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 2 SW Ecosystem & Delivery projects and 4 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 structure reduces complexity of ATDM to ECP coordination and gives ATDM flexibility in revising its portfolio without disruption to the ECP-ATDM mapping.
- Phase 3: 33 total L4 subprojects. Fewer, larger and more uniform-sized projects
  - Starting with FY2020, ECP ST has further consolidated L4 projects to foster additional synergies and amortize project overheads as ECP heads into Critical Decision Phase 2 [7], where more rigor in planning and execution are needed.
  - 5 L3s to 6: New NNSA ST L3
  - 40 ST SC-funded L4 subprojects to 30.
    - Programming Models & Runtimes– 13 to 9, Development Tools- 6 to 6, Mathematical Libraries- 7 to 6, Data & Visualization- 10 to 7, SW Ecosystem & Delivery- 4 to 3.
    - Includes 2 new L4 subprojects in SW Ecosystem.
  - 15 ST NNSA-funded projects transferred to new NNSA ST L3. Consolidated from 15 to 3 L4 subprojects.
  - No more small subprojects.
  - Figure 2 show the overall structure.

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

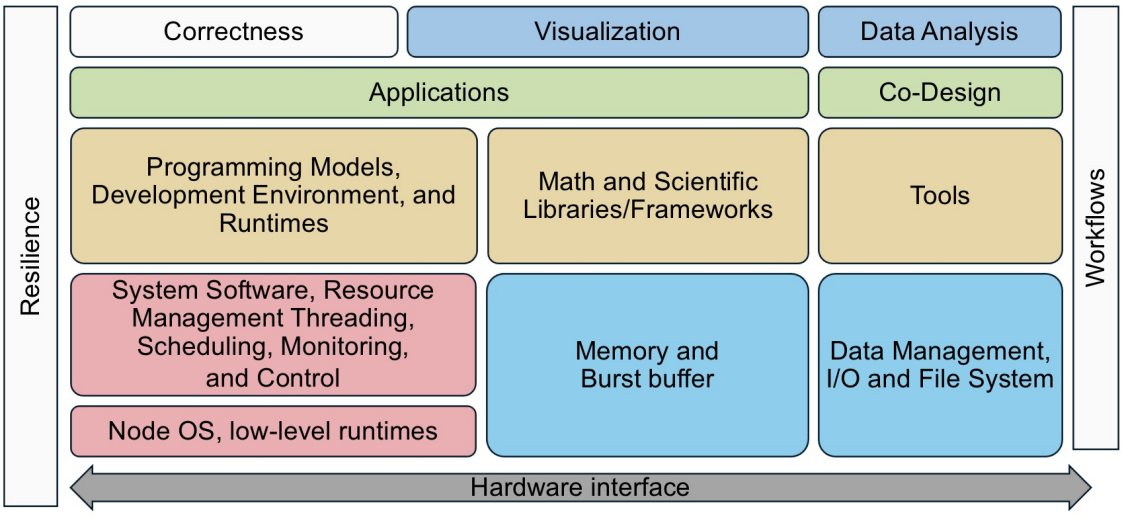


Figure 4: ECP ST before November 2017 reorganization. This conceptually layout emerged from several years of Exascale planning, conducted primarily within the DOE Office of Advanced Scientific Computing Research (ASCR). After a significant restructuring of ECP that removed much of the facilities activities and reduced the project timeline from 10 to seven years, and a growing awareness of what risks had diminished, this diagram no longer represented ECP ST efforts accurately.

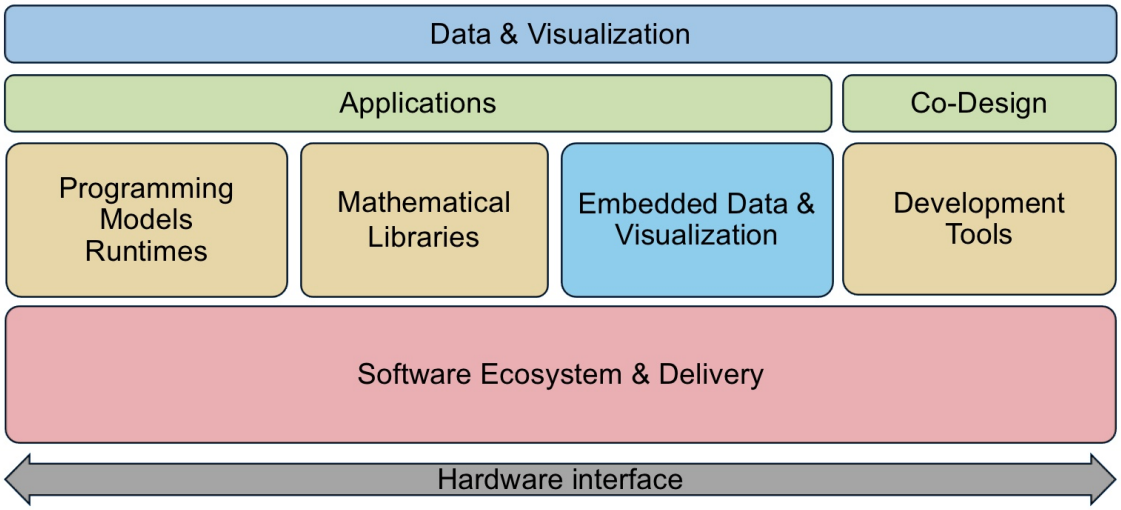


Figure 5: ECP ST after November 2017 reorganization. This diagram more accurately reflects the priorities and efforts of ECP ST given the new ECP project scope and the demands that we foresee.

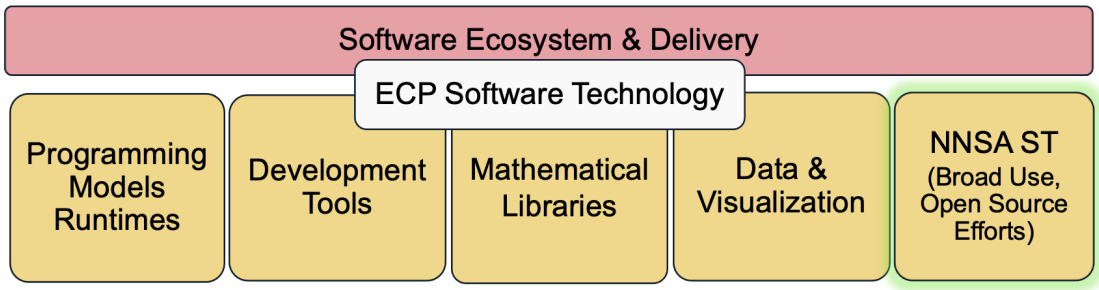


Figure 6: ECP ST after October 2019 reorganization. This diagram reflects the further consolidation of NNSA open source contributions to enable more flexible management of NNSA ST contributions.

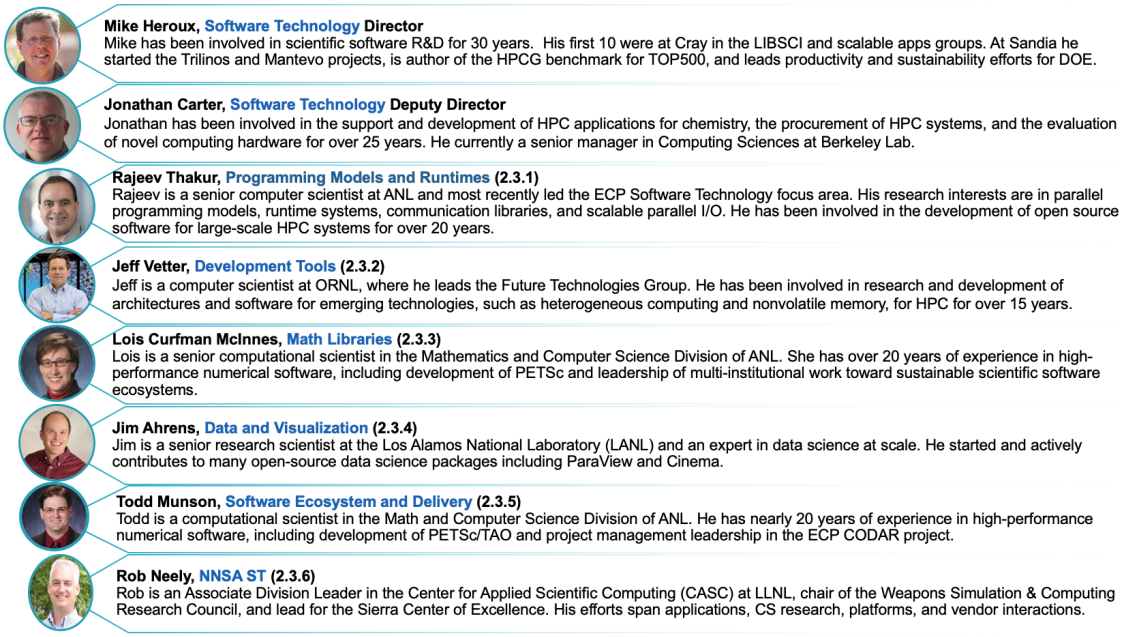


Figure 7: ECP ST Leadership Team as of October 2019.

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 may come from third-party open-source efforts.

The ST portfolio includes both ASCR and NNSA ATDM funded efforts. The MOU 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 is also planning to increase integration within the ST portfolio through increased use of software components and application composition vs. monolithic application design. An important transition that ECP can accelerate is the increased development and delivery of reusable scientific software components and libraries. While 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 an aggregate composition of reusable components than a monolithic code that uses libraries tangentially.

To accelerate this transition, we need a greater commitment on the part of software component developers 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. At the same time, we must expect application developers to participate as clients and users of reusable components, using capabilities from components, transitioning away from (or keeping as a backup option) their own custom capabilities.

2.1.1 The Extreme-scale Scientific Software Stack (E4S)

On November 8, 2018, ECP ST released version 0.1 of the Extreme-scale Scientific Software Stack, E4S (http://e4s.io). It released version 0.2 of E4S in January 2019 and version 1.0 in November 2019. 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 ECP and the broader community. E4S is also the ECP ST vehicle for regression and integration testing across DOE pre-Exascale and Exascale systems.

E4S has the following key features:

- **The E4S suite is a large and growing effort to build and test a comprehensive scientific software ecosystem:** E4S V0.1 contained 25 ECP products. E4S V0.2, release in January 2019 contained 37 ECP products and numerous additional products needed for a complete software environment. Eventually E4S will contain all open source products to which ECP contributes, and all 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 ECP and all additional underlying software required to use ECP ST capabilities. Furthermore, we expect the E4S effort to live beyond the timespan of ECP, becoming a critical element of the scientific software ecosystem.

- **E4S is partitionable:** E4S products are built and tested together using a tree-based hierarchical build process. Because we build and test the entire E4S tree, users can build any subtree of interest, without building the whole stack (see Figure 8).

- **E4S uses Spack:** The Spack [2] meta-build tool invokes the native build process of each product, enabling quick integration of new products, including non-ECP products.

- **E4S is available via containers:** In addition to a build-from-source capability using Spack, E4S maintains several container environments (Docker, Singularity, Shifter, CharlieCloud) that provides 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, the ECP application project CANDLE (Cancer Deep Learning Environment) uses an E4S container to provide a turnkey tutorial execution environment.

- **E4S distribution:** E4S products are available at http://e4s.io.

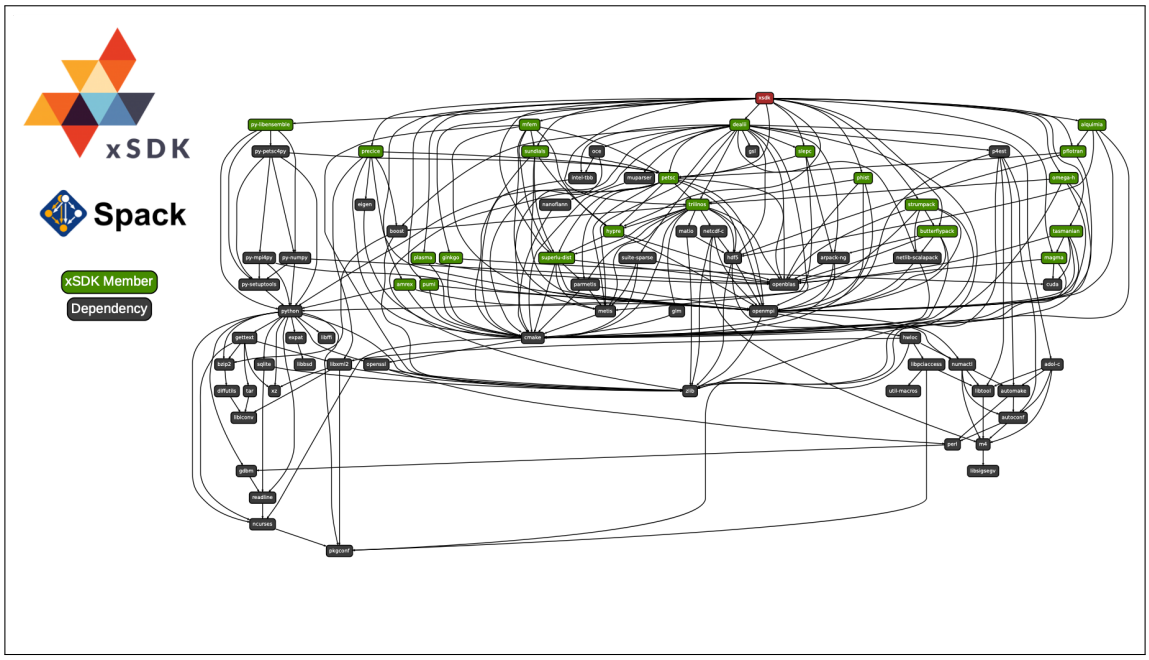


Figure 8: Using Spack [2], E4S builds a comprehensive software stack. As ECP ST efforts proceed, we will use E4S for continuous integration testing, providing developers with rapid feedback on regression errors and providing user facilities with a stable software base as we prepare for Exascale platforms. This diagram shows how E4S builds ECP products via an SDK target (the math libraries SDK called xSDK in this example). The SDK target then builds all product that are part of the SDK (see Figure 70 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, in this case a particular version of Python. Black-labeled products are expected to be previously installed into the environment (a common requirement and easily satisfied). Using this approach, a user who is interested in only SUNDIALS (a particular math library) can be assured that the SUNDIALS build will be possible since it is a portion of what E4S builds and tests.

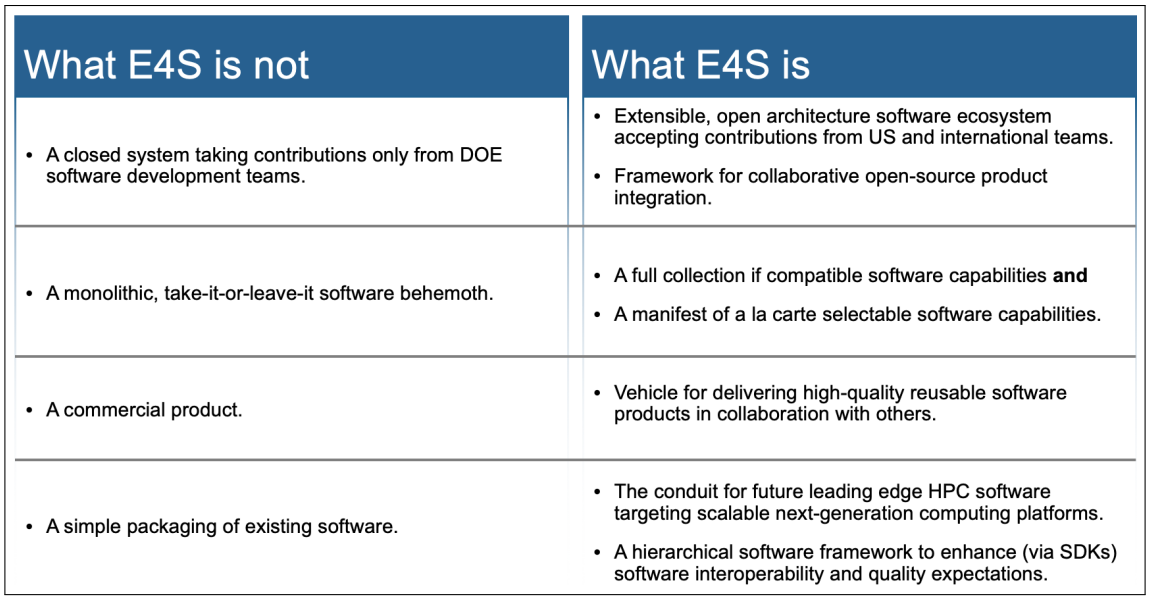


Figure 9: The Extreme-scale Scientific Software Stack (E4S) provides a complete Linux-based software stack that is suitable for many scientific workloads, tutorial and development environments. At the same time, it is an open software architecture that can expand to include any additional and compatible Spack-enabled software capabilities. Since 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.

- **E4S developer community resources:** Developers interested in participating in E4S can visit the E4S-Project GitHub community at `https://github.com/E4S-Project`.

The E4S effort is described in further detail in Sections 4.5, especially Section 2.1.2.

2.1.2 Software Development Kits

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

ECP ST SDKs As part of the delivery of ECP ST capabilities, we will establish and grow a collection of SDKs. The new layer of aggregation that SDKs represent are 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 ECP to include any relevant external effort. The meta-build systems (based on Spack) will play an important role in managing the complexity of building the ECP ST software stack, by providing a new layer where versioning, consistency and build options management can be addressed at a mid-scope, below the global build of ECP ST products. Each ECP ST L3 (five of them) has funds for an SDK project from which we have identified a total of six SDKs and an at-large collection of remaining products that will be delivered outside of the SDK grouping. Section 4.5.7 provides an update on the progress in defining SDK groupings. For visibility, we provide the same diagram in Figure 12.

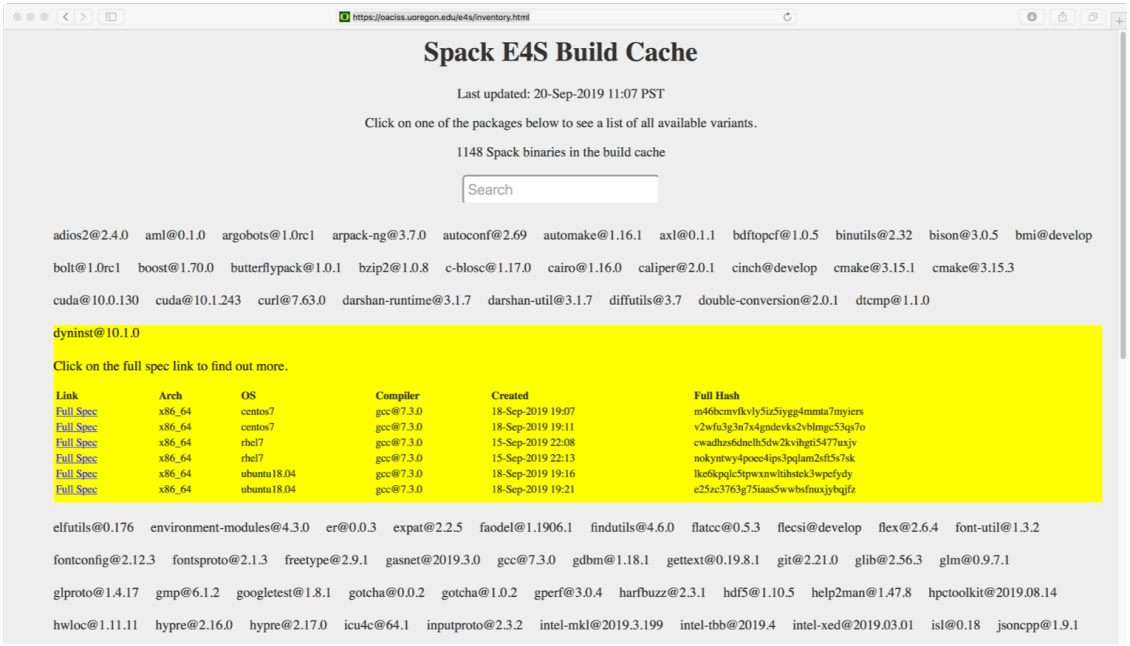


Figure 10: Using Spack build cache features, E4S builds can be accelerated by use of cached binaries for any build signature that Spack has already seen.

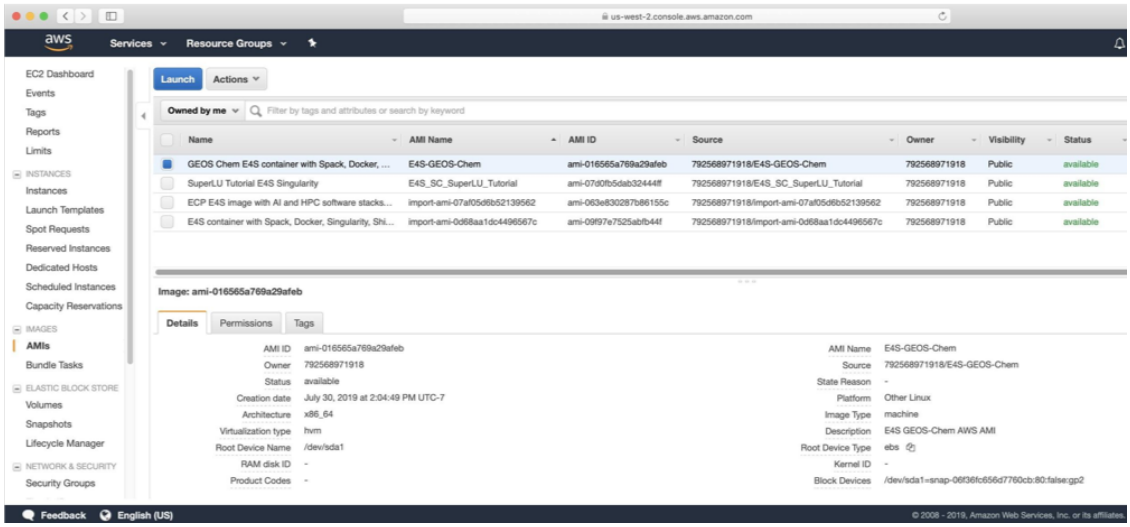


Figure 11: E4S is available as an Amazon AWS public image. Images on Google and Microsoft Cloud environments will be available soon.

1. **Domain scope:** Each SDK will be composed of 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 may have a tight complementary such that ready composability is valuable to the user.
2. **Interaction models:** How packages within an SDK interact with each other. Interactions include common data infrastructure, or seamless integration of other data infrastructures; access to capabilities from one package for use in another.
3. **Community policies:** Expectations for how package teams will conduct activities, the services they provide, software standards they follow, and other practices that can be commonly expected from a package in the SDK.
4. **Meta-build system:** 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 product suite.

Table 2: Software Development Kits (SDKs) provide an aggregation of software products that have complementary or similar attributes. ECP ST uses SDKs to better assure product interoperability and compatibility. SDKs are also essential aggregation points for coordinated planning and testing. SDKs are an integral element of ECP ST [4]. Section 4.5.7 describes the six SDK groupings and the current status of the SDK effort.

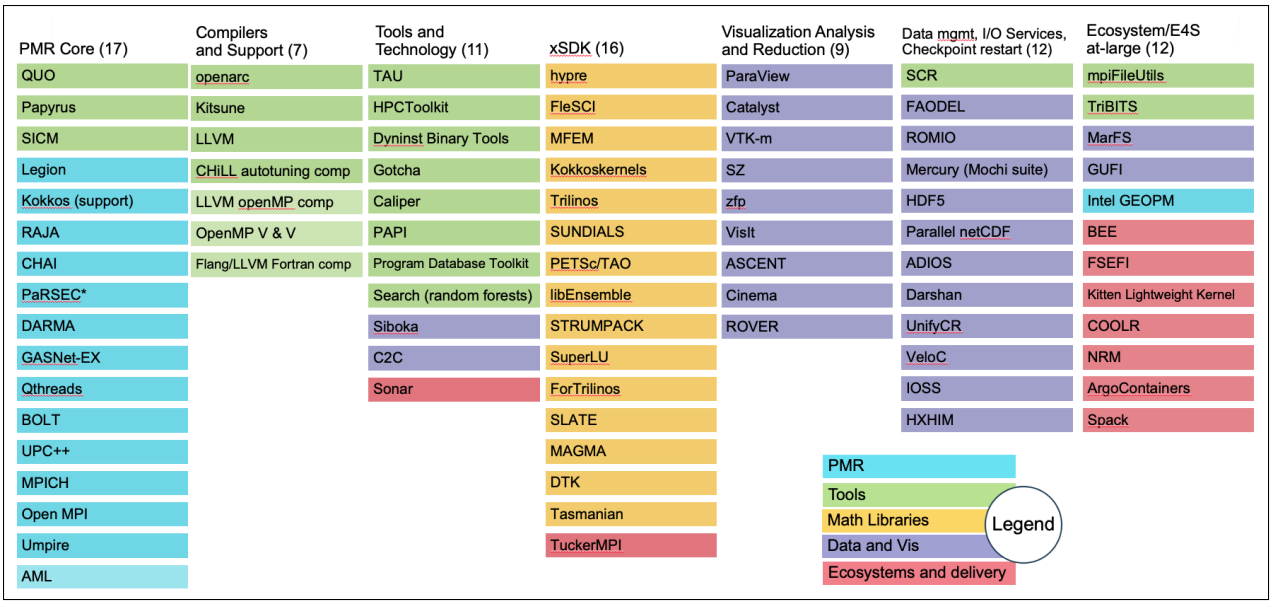


Figure 12: The above graphic shows the breakdown of ECP ST products into 6 SDKs (the first six columns). The rightmost column lists products that are not part of an SDK, but are part of Ecosystem group that will also be delivered as part of E4S. The colors denoted in the key map all of the ST products to the ST technical area they are part of. For example, the xSDK consists of products that are in the Math Libraries Technical area, plus TuckerMPI which is in the Ecosystem and Delivery technical area. Section 4.5.7 provides an update on the progress in defining SDK groupings.

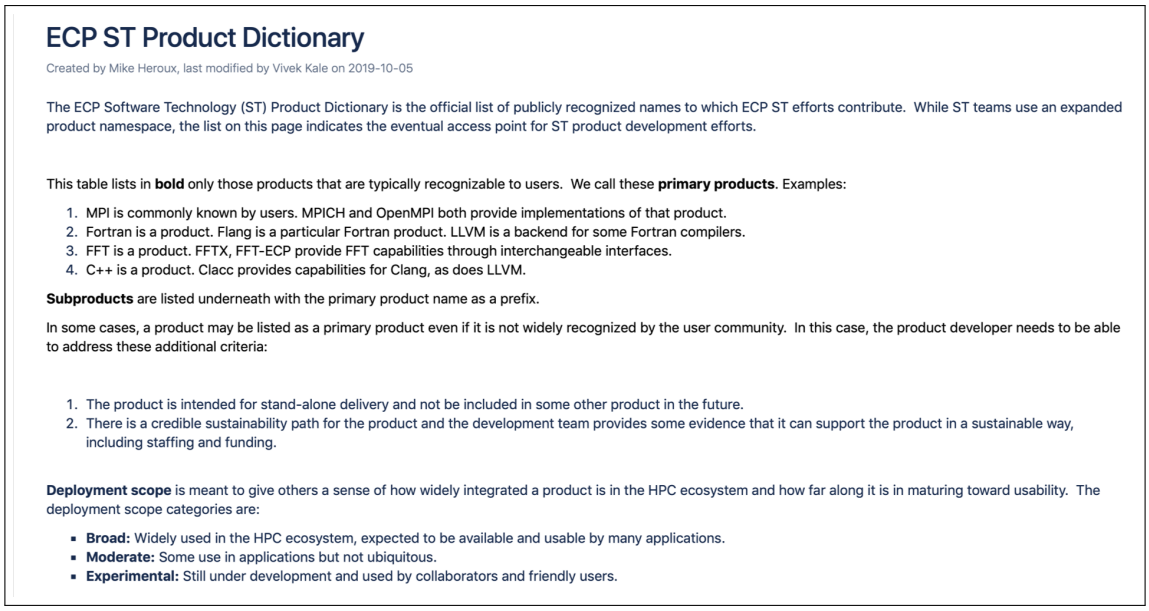
2.1.3 ECP ST Product Dictionary

In the past year, ECP has initiated an effort to explicitly manage ECP ST products and their dependencies (see Section 2.1.4). In order to eliminate ambiguities, we first need a product dictionary: an official list of publicly-name products to which ECP ST teams contribute their capabilities and upon which ECP ST clients depend. The ECP Product Dictionary is single, managed table. It presently contains 70 primary products along with subproducts that are either components within a product or particular implementations if a standard API. Two special primary products are the Facilities stack and Vendor stack. Having these stacks on the list enables ST teams to indicate that their capabilities are being delivered to ecosystems outside of ECP.

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, ECP as a project has created a dependency database that enabled creation and characterization of product-to-product dependencies. ECP manages these dependencies in a Jira database 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 both within and outside the project. Figure



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.

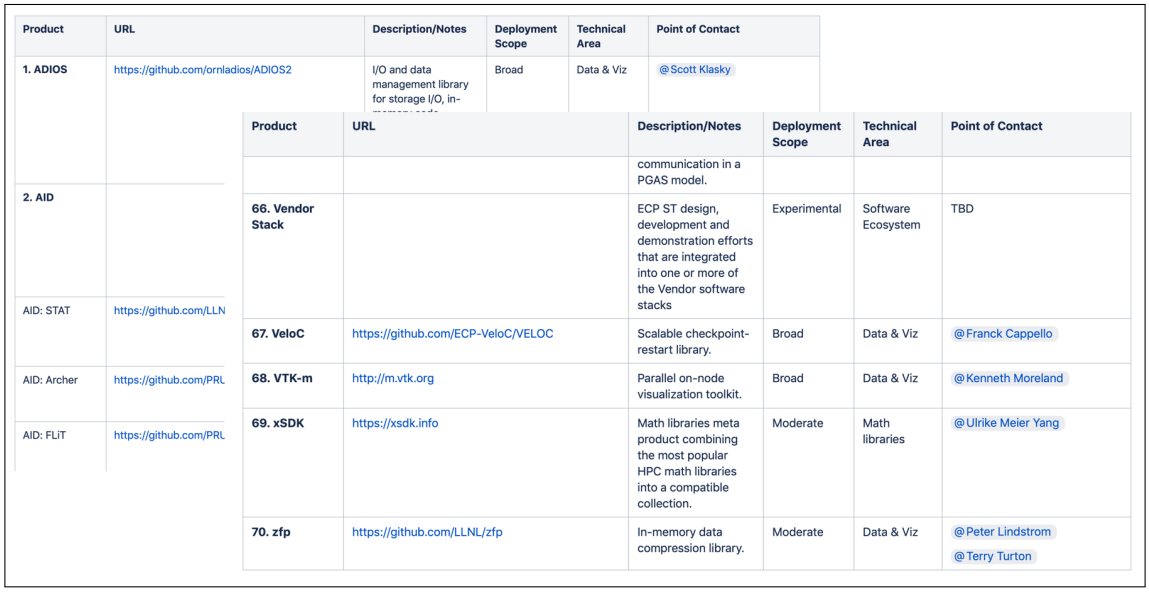
In some cases, a product may be listed as a primary product even if it is not widely recognized by the user community. In this case, the product developer needs to be able to address these additional criteria:

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: This figure shows a 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 (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
		communication in a PGAS model.			
2. AID					
AID: STAT	https://github.com/LLN				
AID: Archer	https://github.com/PRL				
AID: FLiT	https://github.com/PRL				

Product	URL	Description/Notes	Deployment Scope	Technical Area	Point of Contact
66. Vendor Stack		ECP ST design, development and demonstration efforts that are integrated into one or more of the Vendor software stacks	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: These screen shots are from the ECP Confluence Product Dictionary Table. The table is actively managed to include primary and secondary products to which ECP ST team contribute 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 listed primary product in this figure. 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.

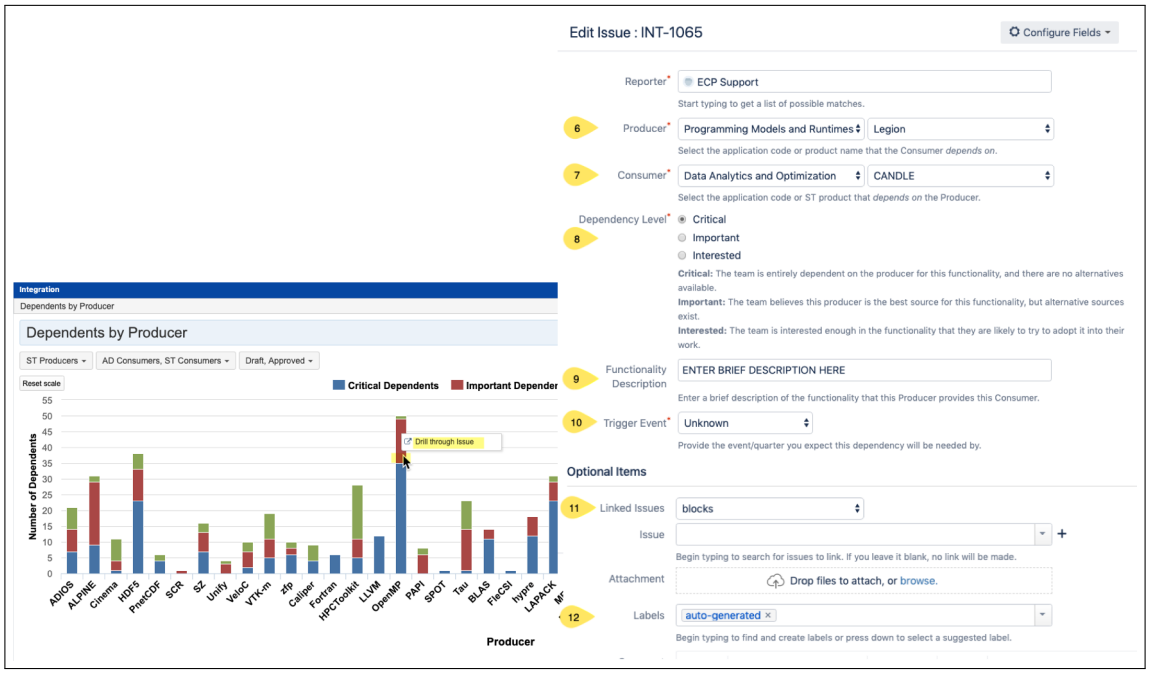


Figure 15: Using Jira, ECP manages its AD, ST, HI, vendor and facilities dependencies. This figure shows a dashboard snapshot along with an edit panel that support creation and management of a consumer-on-producer dependency.

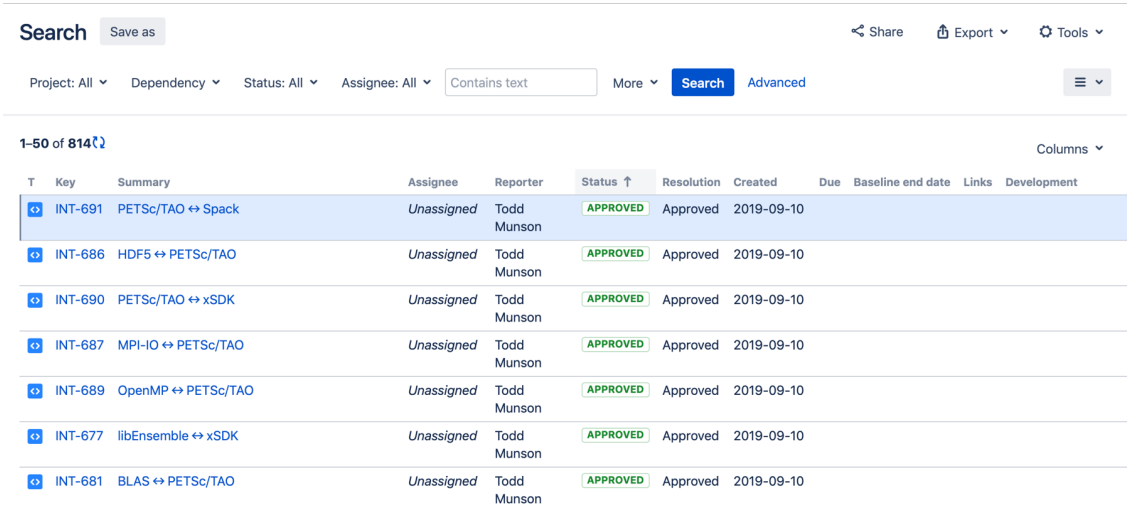


Figure 16: This query result from the ECP Jira Dependency database lists all consumers of capabilities from the PETSc/TAO product. By selecting the details of one of the dependency issues, one can further see how critical the dependency is and see any custom information peculiar to the particular dependency.

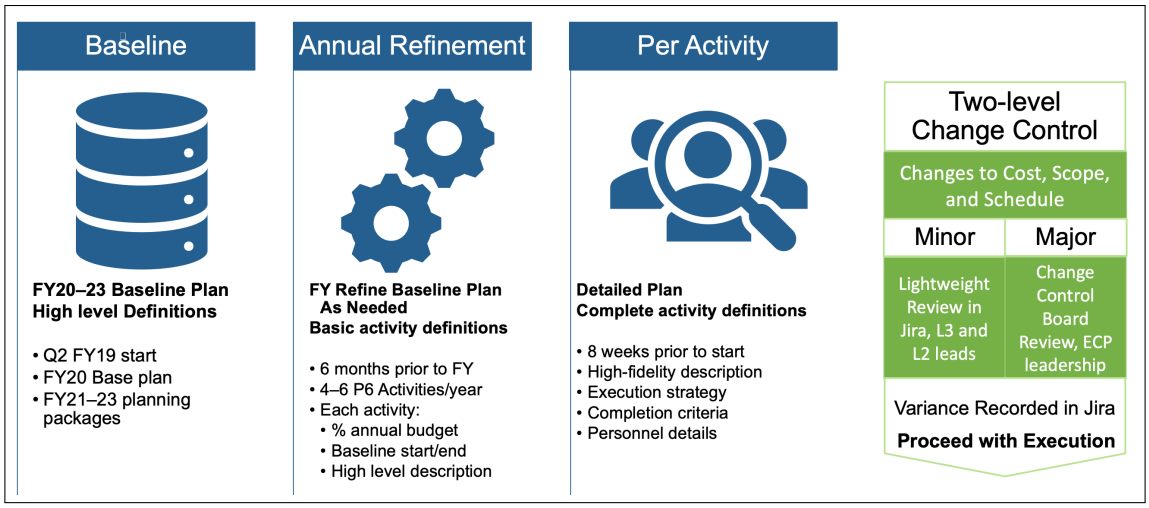


Figure 17: ECP ST uses a custom Jira issue type called P6 Activity. Each L4 subproject creates a series of these issues extending to the end of ECP. Except for the current fiscal year, a single P6 Activity issue describes expected deliverables as a planning package. Six months prior to the start of a fiscal year, the planning package is replaced with 4–6 issues spanning the coming year. Eight weeks prior to the start of an activity, full details about staffing, completion criteria and more are added to the issue.

2.2 ECP ST PLANNING AND TRACKING

While ECP is an official 413.3b federal construction project using an earned value management (EVM) structure, we are permitted to tailor the planning process in order to obtain the flexibility needed for a software project whose requirements are emerging as the project proceeds. In this section, we describe how ECP ST plans it activities using the Jira project management tool. We first discuss P6 Activities (similar to milestones) and then discuss the key performance parameter (KPP-3) associated with 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 extending to the end of ECP (Q3FY23). Except for the current fiscal year, a single P6 Activity issue describes expected deliverables as a planning package. Six months prior to the start of a fiscal year, the planning package for the coming year is replaced with 4–6 issues spanning the year with baseline start and end dates, an estimate of the percent annual budget and a high level description. Eight weeks prior to the start of an activity, full details about staffing, completion criteria and more are added to the issue. Figure 17 show the steps in diagram form.

Cost, scope and schedule for ECP ST is tracked and managed by monitoring 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 our P6 Activities. Schedule, cost and scope changes against the plan are managed via a formal project change request (PCR) process.

2.2.2 Key Performance Parameter (KPP) 3

ECP has four Key Performance Parameters (KPPs). Figure 18 shows the KPP definitions. KPP-3 is focused on a productive and sustainable software ecosystem. ECP ST is the primary owner of this KPP (along with co-design projects in ECP AD). The focus of KPP-3 is defining and tracking capability integrations of ST products into client environment, as described in this section.

First, we define terms:

KPP ID	Description of Scope	Threshold KPP	Objective KPP	Verification Action/Evidence
KPP-1	Performance of scientific and national security applications relative to today's performance	50% of selected applications achieve Figure of merit* improvement ≥50	100% of selected applications achieve Figure of merit improvement 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**	**Software teams meet 50% of their weighted impact goals***	**Software teams meet 100% of their weighted impact stretch goals**	**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 assessment of the impact and timeliness of PathForward milestones

Figure 18: ECP has four key performance parameters (KPPs). ECP ST is the primary owner (with ECP AD co-design subprojects) of KPP-3.

- **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 where 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.
- **Sustainable:** For the purposes of KPP-3, sustainable means that the capability is integrated in a way that reasonably assures future use of the capability beyond the end of ECP. For libraries, this would generally mean that library usage is made from source code in the main repository and use of the library is tested regularly as a part of the client code regular regression testing. For tools, sustainable would generally mean 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 a subject matter expert.

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

- 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 into whose environments the product capabilities will be integrated. Specific clients can be listed, but are not necessary. Clients must be part of 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, addressing what it means to be successfully integrated.

Demonstration and recording of progress toward 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 4 summarizes the defined values.

Integration Score	Capability	Integration Description
1 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).	Client acknowledges 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 3: Integration Goal Scoring: A 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 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, 4 for larger integration efforts, 8 for smaller ones. This is equivalent to accomplishing one to two capability integration 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 allows us to see the overall integration potential.

Table 4: Key metric values: These values are determined by the L4 sub-project team when defining their KPP-3 issue.

Assessment process While progress is recorded as it is achieved, progress assessment is done annually, including input from external subject matter experts (SMEs). ECP leadership and SMEs will review integration score evidence, confirming or adjusting integration scores. Note: Assessment 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 by a variety of means such that ECP Leadership and SMEs are reasonably certain 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. Note: The planned exascale system (El Capitan) that can support National Security applications will not be available until the end of FY23. Integration of ST products into National Security Applications will be considered for transition from tentative to confirmed when either a) 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 b) integration is demonstrated on the El Capitan early access systems, and exercises capabilities similar to those anticipated to be important to effectively using El Capitan. For KPP-3 capability integrations targeted at El Capitan, we will use the best available confirmation process in FY23. KPP-3 weighted scoring

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

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 one.
Risk-Mitigating	0.5	Some KPP-3 Jira issues are associated with products that help us plan for the potential risks if high impact products don't deliver as expected.
Shared	0.5	Some projects receive funding from both NNSA and SC, e.g. RAJA/Kokkos. For these projects, the score is balanced to reflect dual contributions.

Table 5: 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

An 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 high-performance computing community. The ECP ST Software Ecosystem and Delivery sub-element (WBS 2.3.5) and the SDKs in each other sub-element provide the means by which ECP ST will deliver its capabilities.

ECP ST Delivery and HI Deployment Providing the ECP ST software stack to ECP applications requires coordination between ECP ST and ECP HI. The focus areas have a complementary arrangement where ECP ST delivers its products and ECP HI deploys them. Specifically:

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

Separating the concerns of delivery and deployment is essential because these activities require different skill sets. Furthermore, 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. In addition, ECP HI, the computer system vendors and other parties provide deployable software outside the scope of ECP ST, therefore having the critical mass of skills to deploy the entire software stack.

ECP ST Delivery Strategy ECP ST delivers it software products as source code, primarily in repositories found on GitHub, Gitlab installations or similar platforms. Clients such as ECP HI, OpenHPC and application developers with direct repository access then take the source and build, install and test our software. The delivery strategy is outlined in Figure 19.

Users access ECP ST products using these basic mechanisms:

- **Build from source code:** The vast majority of ECP ST products reach at least some of their user base via direct source code download from the product repository. In some cases, the user will download a single compressed file containing 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, the user executes a configuration process that detects local compilers and libraries and then builds the product. This kind of access can represent a barrier for some users, since the user needs 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 source can be a preferred approach for users who want control over compiler settings, or want to adapt how

the product is used, for example, turning on or off optional features, or creating adaptations that extend product capabilities. For example, large library frameworks such as PETSc and Trilinos have many tunable features that can benefit from the user building from source code. Furthermore, these frameworks support user-defined functional extensions that are easier to support when the user builds the product from source. ECP ST is leveraging and contributing to the development of Spack [1]. Via meta-data 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 multi-product builds.

- **DOE computing facilities:** Each DOE computing facility (ALCF, OLCF, NERSC, LLNL and ACES [LANL/SNL]) provides pre-built versions of 17 to 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, e.g., Flang (Section 4.2.20) and LLVM (Section 4.2.17), and parallel programming environments such as MPICH (Section 4.1.8), OpenMPI (Section 4.1.14) and OpenMP (Section 4.2.19). Development tools such as PAPI (Section 4.2.8) and TAU (Section 4.2.15) are often part of this suite, if not already included in the vendor stack. Math and data libraries such as PETSc (Section 4.3.8), Trilinos (Section 4.3.16), HDF5 (Section 4.4.13) and others are also available in some facilities software installations. We anticipate and hope for increased collaboration with facilities via the ECP Hardware & Integration (HI) Focus Area. We are also encouraged by multi-lab efforts such as the Tri-Lab Operating System Stack (TOSS) [8] that are focused on improving uniformity of software stacks across facilities.

- **Vendor stacks:** Computer system vendors leverage DOE investments in compilers, tools and libraries. Of particular note are the wide use of MPICH(Section 4.1.8) as 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, in particular via OpenHPC[9]. ECP ST intends to foster 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.

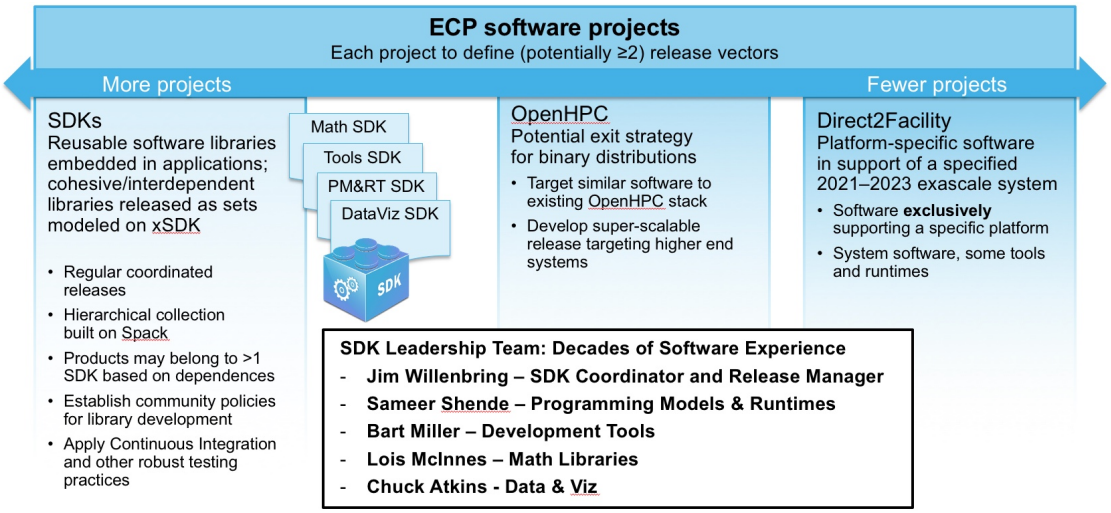


Figure 19: The ECP ST software stack is delivered to the user community through several channels. Key channels are via source code, increasingly using SDKs, direct to Facilities in collaboration with ECP HI, via binary distributions, in particular the OpenHPC project and via HPC vendors. The SDK leadership team includes ECP ST team members with decades of experience delivering scientific software products.

3. ECP ST DELIVERABLES

ECP ST efforts contribute to the HPC software ecosystem in a variety of ways. Most tangible are the contributions to software products, many of which are already widely deployed and being transformed for use with Exascale systems. However, 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 informs the implementation of vendor and other third-party software products. While 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 in five technical areas (Table 1). In each table is a list of related projects, a URL (if available) and an estimate of deployment scope.

3.2 STANDARDS COMMITTEES

An important activity for ECP ST staff is participation in standards efforts. In many instances, our software will not be sustainable if it is not tightly connected to a standard. At the same time, any standard has to take into account the emerging requirements that Exascale platforms need in order to achieve performance and portability. Figure 20 summarized ECP ST staff involvement in the major standards efforts that impact ECP.

ECP ST staff are heavily involved in MPI and OpenMP standards efforts. ECP ST staff hold several key leadership positions and have heavy involvement in all aspects. ECP ST staff also play a critical role

Product	Website	Deployment Scope
GASNet-EX	http://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
Legion	http://legion.stanford.edu	Moderate
Qthreads	https://github.com/Qthreads	Moderate
Umpire	https://github.com/LLNL/Umpire	Moderate
UPC++	http://upcxx.lbl.gov	Moderate
UMap	https://github.com/LLNL/umap	Moderate
BOLT	https://github.com/pmodels/bolt	Experimental
Argobots	https://github.com/pmodels/argobots	Experimental
Intel GEOPM	https://geopm.github.io	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 6: Programming Models and Runtimes Projects (17 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://www.cs.uoregon.edu/research/tau	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 (PDT)	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
QUO	https://github.com/lanl/libquo	Experimental
SICM		Experimental
SuRF		Experimental

Table 7: Development Tools Projects (22 total).

Product	Website	Deployment Scope
hypre	http://www.llnl.gov/casc/hypre	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://computation.llnl.gov/projects/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 8: Mathematical Libraries Projects (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/gsjaardema/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
VeloC	https://xgitlab.cels.anl.gov/ecp-veloc	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
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		Experimental
Siboka		Experimental
SZ	https://github.com/disheng222/SZ	Experimental
UnifyFS	https://github.com/LLNL/UnifyFS	Experimental

Table 9: Visualization and Data Projects (26 total).

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

Table 10: Software Delivery and Ecosystems Projects (2 total).

in C++ standards efforts. While DOE staff have only recently engaged in C++ standards, our efforts are essential to getting HPC requirements considered, especially by contributing working code that demonstrates requirements and design. ECP ST sponsors the newest open source Fortran compiler Flang 4.2.20, a front end for LLVM. This compiler is a rapidly emerging and essential part of the HPC ecosystem. In particular, while 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, including the *de facto* also provide valuable leverage and improved uniformity for HPC software. Lastly, we mention the Visualization Toolkit (VTK) Architecture Review Board (ARB). While this is only a single instance, we intend to explore the ARB model as part of our SDK efforts.

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

Figure 20: ECP ST staff are involved in a variety of official and *de facto* standards committees. Involvement in standards efforts is essential to assuring the sustainability of our products and to assure that emerging Exascale requirements are addressed by these standards.

3.3 CONTRIBUTIONS TO EXTERNAL SOFTWARE PRODUCTS

While much of ECP ST efforts and focus are on the product that we develop and support, it is important to note that some of our important work, and certainly some of our most sustainable and highly leveraged work, is done by providing requirements, analysis, design and prototype capabilities for vendor and other third party software. Many software studies have shown that 70 to 80% of the cost of a successful software product goes into post-delivery maintenance. Our effort summarized in Table 11 expressly eliminate this large cost for DOE because the product is developed and supported outside of 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, effort that require a constant involvement with the other authors, as well as participation to the online discussions related to the chapter and regular attendance of the MPI Forum face-to-face activities.
Flang	ECP funds development of the new open source Fortran compiler front end called Flang. Flang provides Fortran language support for LLVM backends, in a similar way as Clang provides support for C and C++.
All Development Toolswork	Starting in FY20, our Development Tools efforts are organized around delivering capabilities into the LLVM ecosystem.
SWIG (www.swig.org)	The ECP ST ForTrilinos efforts contributes the capability to generate automatic Fortran bindings from C++ code.
TotalView debugger	ECP ST staff are engaged in co-design of OMPD, the new debugging interface for OpenMP programs, along with RogueWave engineers. This effort helps RogueWave improve their 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 IR and loop-optimization infrastructure.
SLATE	ECP ST math libraries 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 IO 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 11: 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 we can do.

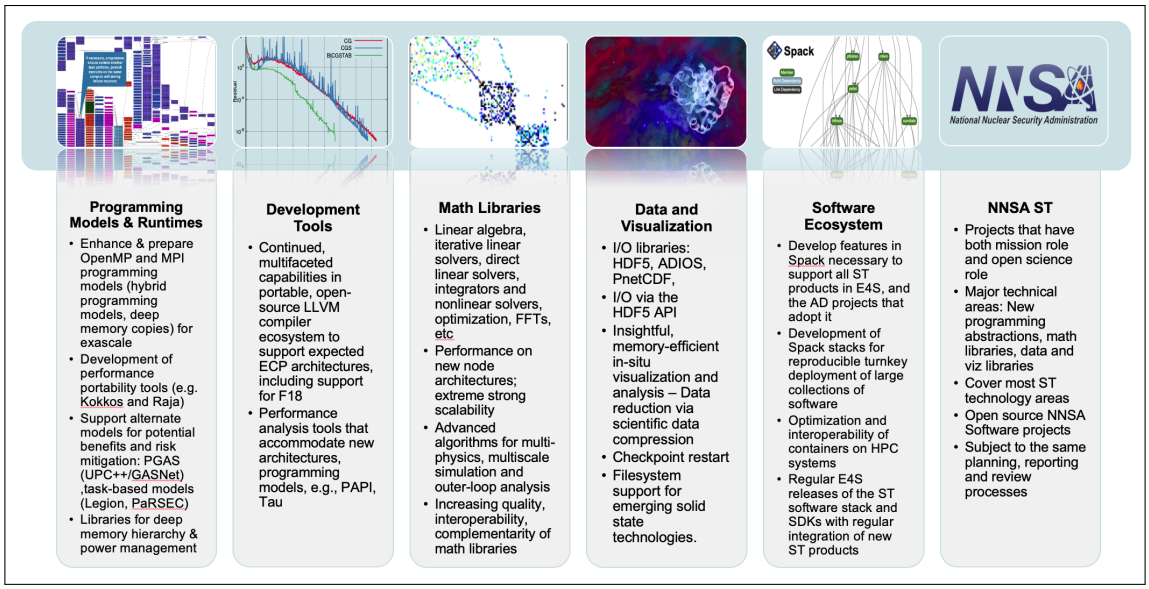


Figure 21: ECP ST is composed of 6 Level-3 Technical Areas. The first four areas are organized around functionality development themes. The fifth is focused on technology for packaging and delivery of capabilities. The sixth is organized around per-lab open source development at the three DOE NNSA laboratories, LANL, LLNL and SNL.

4. ECP ST TECHNICAL AREAS

ECP ST is composed of six Level-3 Technical Areas (see Figure 21). In this section of the ECP ST Capabilities Assessment Report we provide an overview of each Level-3 Technical Area and two-page summaries of each funded project within the technical area. For each L3 area, we discuss scope and requirements, assumptions and feasibility, objectives, plans, risks and mitigations and future trends. For each Level-4 subproject, we provide a project overview and summarizes the key challenges, solution strategy, recent progress and next steps for the project.

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 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.

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 SW 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.

MPICH team is involved in the formation of the MPI Forum and have 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 MVAPICH). MPICH and its derivatives are exclusively used in 7 of the top 10 supercomputers in the world today. 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:

1. Interoperability with intranode programming models having a high thread count [10, 11, 12] (such as OpenMP, OpenACC and emerging asynchronous task models);
2. Scalability and performance over complex architectures [13, 14, 12, 15] (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 [16]. Apart from this, we also looked at communication strategies which allow the MPI library to be as lightweight as possible [17, 18]. 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 [19, 20, 21] started with the survey on heterogeneous memory architectures on upcoming DOE machines and how MPICH can take advantage of them [22]. The efforts also included the investigation of utilizing heterogeneous memory inside the MPI implementation and evaluation of applications [23]. 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 [24, 25] originated with the investigation and evaluation of hints based on topology awareness and optimizations to virtual topology functionality in MPICH [26, 27]. The

other efforts include investigation of topology-aware collectives and neighborhood collectives in MPICH [28] and evaluation of the selected ECP applications.

Exascale MPI fault tolerance efforts [29, 12] 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 (ULFM). Other efforts in this direction includes standardizing ULFM in MPI and evaluating application suitability for fault tolerance.

Exascale MPI+X hybrid programming developed firstly with effort in improving interoperation of MPICH with threads [30]. Secondly, we developed the work-queue data transfer model for multithreaded MPI communication [31]. We have included support for interaction of MPICH with user-level thread (ULT) libraries [32], primarily targeting Argobots and the BOLT runtime [3]. Other issues that are being looked at include the investigation and evaluation on interaction between MPI and OpenMP and the study and evaluation of MPI endpoints.

Recent Progress Figure 22 provides the details of major milestones completed in FY2019. In the first milestone, we studied the performance of the RMA improvements using the large quantum chemistry application suite NWChem (version 6.6) with ARMCI-MPI and MPICH on the Cray XC40 supercomputer. The results shown that enabling network hardware atomics with the info hints fully eliminated the performance bottleneck in the Density Functional Theory (DFT) module and improved the performance scalability of NWChem DFT. In the second milestone, we studied the performance impact of ULFM when there is no failure. A study report has been submitted on performance analysis of the ULFM prototype. In the third milestone, we studied the performance of MPI Endpoints. The results shown significant improvement of using MPI endpoints prototype—the multithreaded MPI communication reached a message rate that is close to the case of single-threaded MPI communication. We concluded that exposing the application level parallelism to the MPI through the use of endpoints enables effectively scheduling of the traffic. In such a way, multiple hardware resource can be utilized to reduce the contention between threads and improve performance.

Figure 22: Major MPICH milestones completed in fiscal year 2019

In the fourth milestone, we evaluated the benefit of using topology-aware neighborhood collectives. We used the neighborhood collective integration in the PETSc scalable linear solvers (KSP) component for this evaluation. The results did show benefit, but with a caveat that is the neighborhood collectives incurs significant setup cost which currently overshadowing the communication benefit. The addition of "persistent" collective operations in MPI-4 will allow us to remove the per-operation setup cost. In the last milestone, we performed comprehensive evaluation of the project which included functional tests and performance tests. The experiment results using Nek5000 on OLCF Summit supercomputer shown significant improvements in performance and scalability due to the techniques developed in this ECP project.

Next Steps A major focus of the ongoing Exascale MPI efforts is MPI+GPU improvements. This includes improvements for multiple accelerator nodes and native hardware models, support for noncontiguous data

and software evaluations. Exascale MPI ongoing efforts also includes a developing a collective selection framework for improving the performance and scalability of collectives. We are also making efforts in MPI standardization which includes investigation application usage of MPI and incorporating those insights into the MPI standard through continuous participation of the MPI standardization process.

4.1.9 WBS 2.3.1.08 Legion

Overview This project focuses on the development, hardening and support of the Legion Programming System with a focus on Exascale platforms and workloads that can benefit from Legion's features. At a fundamental level our focus is on the key capabilities (e.g. correctness, performance, scalability) of an alternative programming model for ECP that seeks to expose additional levels of parallelism in applications. 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 are currently focused on addressing bugs, refactoring the implementation for improved stability, performance and scaling, extending support for the selected exascale platforms (Aurora and Frontier), and also expanding the feature set as needed for both application and platform nuances.

The Legion Programming System is freely available with an Apache-based open source license and is hosted at GitLab:

https://gitlab.com/StanfordLegion/legion

Key Challenges While Legion addresses a number of key challenges in improving system utilization and some aspects of platform portability, it is a relatively new programming system and therefore there is a cost to rewriting applications. This aspect makes significant adoption a risk within ECP and additional effort must also take place to catch up with aspects of performance and scaling to match aspects of more mature technologies.

We have recently started to focus much of our efforts on emerging use cases that are related to machine learning and data-centric workloads. These domains are much easier to have a substantial impact as the application codes rely on external tools (e.g. TensorFlow, Python, etc.) vs. years of established code written in MPI and/or OpenMP. We are already seeing clear benefits of focusing our efforts in this direction. This has helped us to increase our overall impact as well focus on areas of adoption across more specialized application needs in support of machine learning and other data-centric workloads.

Solution Strategy As a collaboration between Los Alamos, Stanford University and other efforts at NVIDIA and SLAC, we are providing the overarching implementation Legion programming model that captures the "best" (correct and feature complete) version of the code. In addition, we are actively looking for opportunities to educate the ECP community about Legion and other data-centric and task-based approaches to programming. Most recently we have been closely working with ExaFEL (AD 2.2.4.05) and the CANDLE project (AD 2.2.4.03) to provide support for Legion. We also provide support and software releases related to the efforts going on within LANL's ATDM Programming Models and Runtimes project (part of ST 2.3.6.01), that refine, identify needs and requirements that are in support of Ristra (LANL's National Security application AD 2.2.5.01). Our project includes management of the current repository and quarterly, or more frequent, releases of Legion to the broader community. We are also looking at numerous aspects of having the Legion system interoperate with today's more widely used programming systems – e.g. MPI and OpenMP.

More recently we have started to look at leveraging recent results in improving the performance of training deep learning applications. Our most recent results explore CANDLE's requirements for ML training and inference on large DNNs. This is discussed in more detail below.

Finally, we are actively exploring techniques to simplify Legion programming and improve overall developer productivity. This has been a both a broader community effort that focuses on both connections with Python (e.g. see [33] and [34]) and also furthering the development and performance of FlexFlow [35] by looking at the impact of performance improvements to Legion such as the work in [36].

Recent Progress As discussed above, our progress to date has been focused improving training times for CANDLE's DNN use cases. For example, training a single epoch of the Uno dataset (with 21 million samples) using TensorFlow requires 27 hours on 1 GPU and 18 hours on 3 GPUs (where it reaches a scalability limit for TensorFlow). Using FlexFlow, a deep learning framework built on top of Legion, the training time is reduced to 1.2 hours on 128 Summit nodes, using 768 GPUs. This represents a 15x speedup and as shown in Figure 23 has significantly better scaling than TensorFlow.

In addition to the these efforts we continue to focus on bug fixes, performance improvements and overall scalability. We have also started to address exascale platform-centric details and have started to work on targeting AMD's software infrastructure for Frontier and are beginning to explore the details of Intel's software stack for Aurora. In addition, we have started to diversify and modularize the low-level transport layers of the system to use both GASNetEX (ST 2.3.1.14) and a modern MPI layer (ST 2.3.1.07). This approach will give us multiple transport layers for helping to reduce potential impacts from the different platforms.

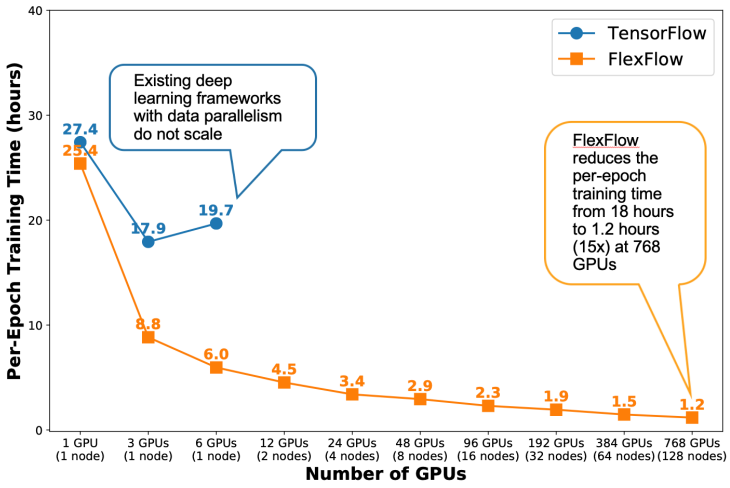


Figure 23: The CANDLE project requires training and inference on large DNNs, which are computationally intensive and difficult to parallelize. Training a single epoch of the Uno dataset (with 21M samples) using TensorFlow requires 27 hours on 1 GPU and 18 hours on 3 GPUs. Using FlexFlow,a deep learning framework built on top Legion, reduces training time to 1.2 hours on 128 Summit nodes (768 GPUs). The features enabling this are Legion's first-class data partitioning, which enables more flexible and efficient parallelization strategies than are supported by existing frameworks.

Next Steps Our plans for the next year are to continue focusing on the challenges presented by the upcoming exasacle system architectures and on hardening and improving the overall performance and scalability of Legion. In addition to the Office of Science systems (Aurora and Frontier) we will also consider support for the Sierra and El Capitan platforms at Lawerence Livermore to support NNSA components that are using Legion. We will continue to seek out and improve our educational outreach and developer productivity, including regular open source releases of Legion and direct interactions with the applications teams for debugging, fine-tuning of features for particular use cases and overall performance tuning. We are also actively interacting with a growing community of interested users from outside the traditional HPC community where data-centric and accelerated machine learning use cases are of growing importance.

4.1.10 WBS 2.3.1.09 Distributed Tasking at Exascale: PaRSEC

Overview The PaRSEC Environment 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 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.

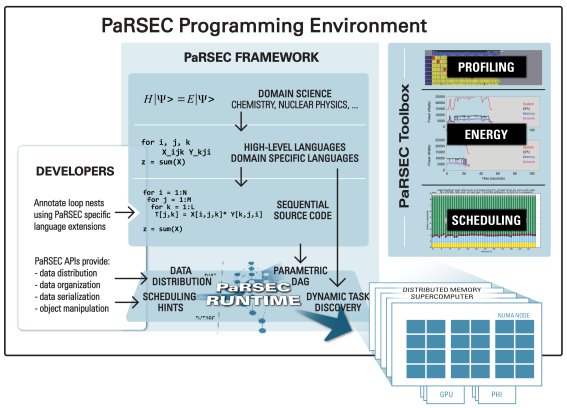


Figure 24: PaRSEC architecture based on a modular framework where each component can be dynamically activated as needed.

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 runtimes, 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, performant and Exascale ready programming paradigms.

Key Challenges As Exascale platforms delivery become a closer deadline, a 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.

Solution Strategy 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 it's capabilities to the hardware and to improve it's 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 [37, 38, 39].

Recent Progress The software release (2019.11) provides many new additions to the low-level task runtime, supports for a number of hardware capabilities (GPU, NVLink, P9 atomic ops), brings significant improvements to the performance and scalability of the runtime, and addresses many pending issues. On

the software quality side, the PaRSEC runtime has been evaluated and amended to compile and run on all pre-Exascale platforms (ALCF Mira, Theta; OLCF Summit). PaRSEC now includes a Spack definition file to ease the deployment on future target systems as part of the system software SDK effort.

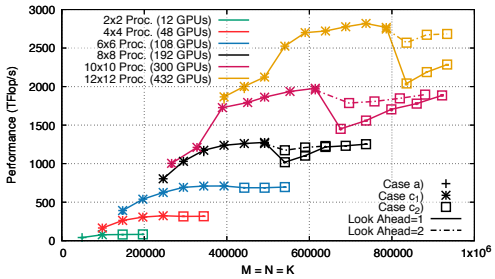


Figure 25: Strong-scaling performance of the matrix-matrix multiplication (PDGEMM)

The DTE GPU support engine has been refactored, addressing some of the existing limitations in data management (allocation and transfer), and adding support for NVLink capabilities to minimize the transfer cost between GPUs on the same node. All these improvements have enabled unprecedented performance on distributed, multi-GPU platforms for real applications. Figure 25 shows a strong scaling performance of PDGEMM from our DPLASMA math library using PaRSEC to power one of the most compute intensive operation and study the implementation scalability with an increasing problem and number of computing elements size. This integration illustrates the runtime capability to obtain high efficiency for such operation independent on the number of resources involved in the computation, even in scenarios where the memory required for the storage of the matrices is larger than the total amount of memory available on the GPU. A pro-active data transfer management system and an efficient data transfer mechanisms are some of the PaRSEC underlying features that enable such level of performance.

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. The PaRSEC development team did the first step toward the integration of their framework into the SLATE (2.3.3.09) in the context of the shared milestone (STPM11-23). The first prototype of the application ran 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.

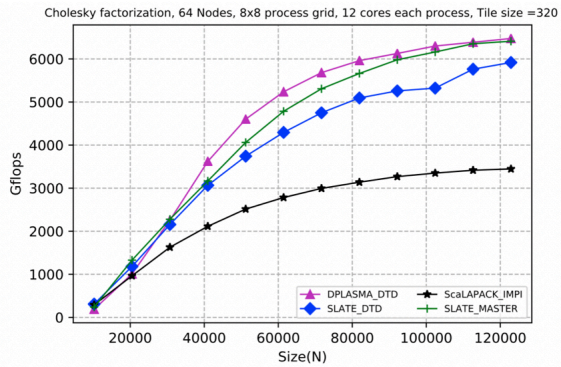


Figure 26: Comparison of DPLASMA and SLATE Cholesky factorization over PaRSEC with SLATE and ScaLAPACK on 64 nodes 12 cores each

In figure 26, we compare the integration of SLATE and PaRSEC against the state of the art. First against the two legacy domain specific languages that have the capability to do linear algebra; then against the regular SLATE using OpenMP for intra-node parallelism, and MPI for communication; and finally against ScaLAPACK, which is the reference for distributed linear algebra.

Next Steps To provide programmers with more supervision 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 The Lightweight Communication and Global Address Space Support project (Pagoda) is developing GASNet-EX [40], 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++ [41, 42] and the Legion Programming System [43, 44] (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, 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 with OpenMP, enabling developers to focus their effort on optimizing performance-critical communication.

We are co-designing GASNet-EX with the UPC++ development team with additional input from the Legion and (non-ECP) Cray Chapel [45, 46] 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 that software needs to minimize the work performed by lightweight cores and avoid the overhead of long, branchy serial code paths; this motivates a requirement for efficient fine-grained communication. These problems 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, 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.

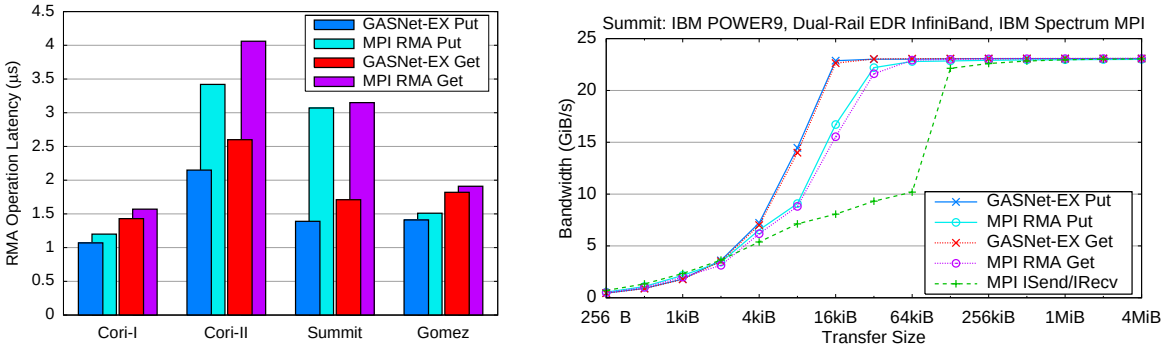
Solution Strategy The PGAS model is a powerful means of addressing these challenges and is critical in building other ECP programming systems, libraries, and applications. We use the term *PGAS* for models that support one-sided communication, including contiguous and non-contiguous remote memory access (RMA) operations such as put/get and atomic updates. Some of these models also include support for remote function invocation. GASNet-EX [47] 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 15 years of experience with the GASNet [48, 40] communication layer to provide production-quality implementations that include improvements motivated by technology trends and application experience.

The goal of the GASNet-EX work is to provide a portable, high-performance GAS 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. These design choices and their implementations supply the low-overhead communications mechanisms required to address the requirements of Exascale applications.

Figure 27 shows representative results from a paper [47] comparing the RMA performance of GASNet-EX with MPI on multiple systems including NERSC's Cori and OLCF's Summit[1]. 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 [40], along with results for additional systems.

Figure 27a shows the latency of 8-byte RMA Put and Get operations on four systems, including two distinct networks 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.

[1]The paper's results from Summitdev have been replaced by more recent (June 2019) results from OLCF's newer Summit system.



(a) 8-byte RMA Latencies

(b) Summit Flood Bandwidth

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

Figure 27b shows flood bandwidth of RMA Put and Get over the dual-rail InfiniBand network of 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 series) illustrates the benefits of one-sided communication, a major feature of PGAS models.

Recent Progress Work on GASNet-EX in the past year has covered several distinct areas.

Co-design work is on-going with the UPC++, Legion and Chapel developers. These interactions are bi-directional, guiding design and implementation decisions made in GASNet-EX as well as in the client runtimes. Recent work [49] for the Cray Aries network has yielded significant performance improvement on systems of importance to UPC++ and Legion. Specifically, implementation of a new target-side reassembly protocol improved AM Long end-to-end latency by up to 33%, and the effective bandwidth by up to 49%, while also enabling asynchronous source completion that drastically reduces injection overheads.

We have begun work to improve interoperability of GASNet-EX and MPI within the same executable. Together with the Exascale MPI project (WBS 2.3.1.07) we have developed an initial implementation of a small library to manage cooperative progress of multiple communications runtimes. We have demonstrated the ability of this library, with suitable calls added to GASNet-EX and MPICH, to resolve two canonical examples of the class of deadlock it is meant to prevent.

We have completed work to leverage features of the InfiniBand network present in OLCF's Summit. One such development is modifications to fully utilize the multiple InfiniBand network ports (a.k.a "rails") in a Summit node. Another is use of Mellanox's "ODP" (On-Demand Paging) to provide efficient and robust RDMA transfers to and from memory in a processes' stack and dynamic heap.

Next Steps Our next efforts include:

1. **Device (GPU) Memory Support**. GASNet-EX's RMA APIs have been designed to enable hardware offload of transfers to and from GPU memories (*e.g.* use of GPUDirect). Near-future work includes implementation of this capability for OLCF's Summit. This work includes adding support for multiple communications endpoints and multiple memory segments, which also provide benefit to multi-threaded runtimes by reducing contention for shared resources.

2. **Specialization for InfiniBand**. Network-specific implementations of new GASNet-EX features for the InfiniBand network will provide performance benefits on systems such as OLCF's Summit. The benefits of such specialization for the Cray Aries network has been demonstrated previously [50].

3. **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 bottlenecks or performance anomalies which are discovered.

4.1.12 WBS 2.3.1.14 UPC++

Overview The UPC++ project [42] at LBNL is developing a C++ library that supports Partitioned Global Address Space (PGAS) programming [41, 51]. The current ECP-funded version of UPC++ is markedly different from an earlier prototype designated V0.1 [52]. First, all communication is *asynchronous*, to allow the overlap of computation and communication, and to encourage programmers to avoid global synchronization. Second, all communication is *syntactically explicit*, to encourage programmers to consider the costs of communication. Third, UPC++ encourages the use of *scalable data-structures* and avoids non-scalable library features. All of these principles are intended to 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 that runs at close to hardware speeds, embracing interest by vendors in the PGAS model that efficiently matches the RDMA mechanisms offered by network hardware and on-chip communication between distinct address spaces. Because ECP applications rely on irregular representations to improve accuracy and conserve memory, the UPC++ library provides an essential ingredient for the ECP software stack. It enables effective scaling of Exascale software by minimizing the work funneled to lightweight cores, avoiding the overhead of long, branchy serial code paths, and supporting efficient fine-grained communication. 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 load balancing. UPC++'s low-overhead communication mechanisms can maximize injection rate and network utilization, tolerate latency through overlap, streamline unpredictable communication events, minimize synchronization, and efficiently support small- to medium-sized messages arising in such applications. UPC++ enables the ECP software stack to exploit the best-available communication mechanisms, including novel features being developed by vendors. This library offers a complementary, yet interoperable, approach to MPI with OpenMP, 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. To reduce software overheads, UPC++ takes advantage of the GASNet-EX [47, 40] communication library's low-overhead communication as well as access to any special hardware (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:

1. **Increased performance through reduced communication costs:** The UPC++ programmer can expect communication to run at close to hardware speeds. Asynchronous execution enables an application to hide communication behind available computation.

2. **Improved productivity:** UPC++'s treatment of asynchronous execution relies on futures and promises, and these simplify the management of asynchrony.

The PGAS one-sided RMA communication employed by UPC++ benefits application performance by mapping tightly onto the RDMA mechanisms supported by the network hardware. GASNet-EX provides the thin middleware needed to enable this model to run at close to hardware speeds, across platforms ranging from laptops to supercomputers. One-sided communication also has another benefit: it decouples synchronization from data motion, avoiding the fine-grained synchronization overheads of two-sided message-passing.

UPC++'s Remote Procedure Call (RPC) enables the programmer to execute procedure calls on remote processors. 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 problems we have implemented a distributed hash table, which serves as a proxy for a key phase in the HipMer application of the Exabiome Project

(WBS 2.2.4.04). This implementation scales efficiently to a large number of processors. RPC was observed to simplify the implementation considerably, by avoiding data hazards without the need for locking. Figure 28 illustrates the benefits of the UPC++ model in a weak scaling study up to 34,816 processes on the KNL partition of NERSC's Cori.

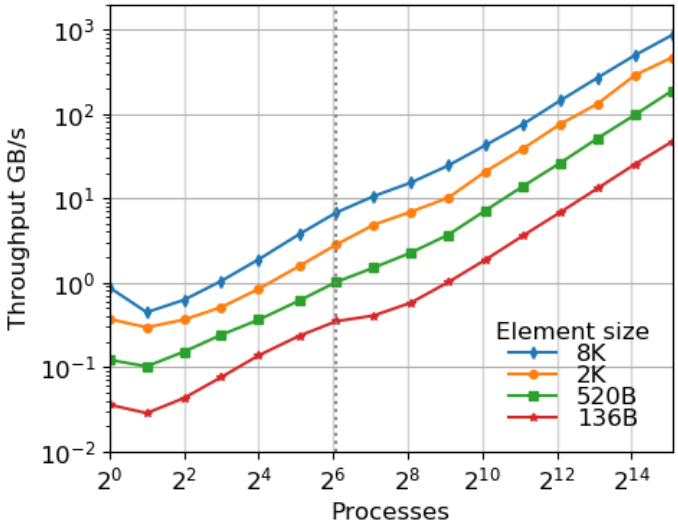


Figure 28: Weak scaling of distributed hash table insertion on the KNL partition of NERSC's Cori platform. The dotted line represents the processes in one node.

Recent Progress

1. **Memory Kinds.** Unified abstractions for transferring data back and forth between device (e.g. GPU) and host memory, possibly remote. By unifying the means of expressing data transfer among the collective memories of a heterogeneous system with different memory kinds, the abstractions enable ECP applications to utilize accelerators, which are necessary to attain peak performance on ECP platforms. The design and abstraction enables eventual hardware offload (such as to GPUDirect) of device data transfers, support for which will be provided in the near future by GASNet-EX.

2. **Portability and Sustainability.** UPC++ benefits to productivity rely heavily on template meta-programming, using features added in C++11. However not all relevant C++ compilers comply sufficiently with the C++11 standard, necessitating deployment of various work-arounds within the UPC++ implementation. The past year has seen the addition of UPC++ support for seven new CPU/compiler pairs. A UPC++ Spack package has also been added to the E4S PMR SDK (Section 2.1.1).

Next Steps

1. **Performance.** With the help of our stakeholders, we are identifying portions of the UPC++ implementation where performance tuning is most needed and/or beneficial. A recent example is deployment of an optimized implementation for RMA puts using target-side completion notification.

2. **Productivity.** With the core API specification and implementation of UPC++ nearly complete, we are shifting focus toward some productivity-oriented features. These include UPC++ serialization of non-trivial user types and a distributed array facility for UPC++.

3. **Outreach.** We have begun activities designed to strengthen collaboration with our stakeholders (both current ones, and potential future ones). This includes holding training events for users of UPC++ and circulating working group drafts of productivity features (above) 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. The problem we are addressing is how to program the deluge of existing and emerging complex memory technologies on HPC systems. This includes the MCDRAM (on Intel Knights Landing), NV-DIMM, PCI-E NVM, SATA NVM, 3D stacked memory, PCM, memristor, and 3Dxpoint. Also, near node technologies, such as PCI-switch accessible 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. Our 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 high-level interface is designed for application developers who would rather define coarser-level constraints on the types of memories the application needs and leave out the details of the memory management. 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 absorbtion into existing open source projects such as hwloc, Umpire, CLANG/LLVM, OpenMP, and Jemalloc.

- Low-Level Interface: Finished refactor of low-level interface supporting memory arenas on different memory types. Added initial support for Umpire, OpenMP. Reviewing features need to fully support these runtimes. 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 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, low_lat) _mem_spaces and supports KNL, Optane, testing on Sierra/Summit.

- High-Level Graph Interface: Metall is 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.

- Analysis: 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.

Next Steps

- Low-Level Interface: Focus on performance of support for runtimes and adding feature requested to support Umpire, OpenMP and MPI and address the slow move pages implementation in the Linux kernel – (collaboration with RIKEN). Test with proxy applications for functionality and correctness. Investigate Linux kernel modifications for page migration in collaboration with ECP project Argo 2.3.5.05 and RIKEN research center in Japan, on-going. Start collaborating with applications to enable use of heterogenous memory on ECP target platforms. Additionally, the team needs to finalize the memory topology discover with the hwloc team.

- For the analysis work the team will extend and hardend the tools for guiding application memory management and investigate feature categories to classify objects associated with different features such as size, type, allocation time, etc to guide data placement.
- For the Metall high-level interface we plan to continue outreach to ExaGraph to store graph data as well as other intermediate data into PM leveraging Metall. We also plan to support UMap (user-level mmap library in Argo PowerSteering project) underneath Metall to enhance its performance and capability.

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% [53]) and the majority of software technology (ST) projects (57% [53]) 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 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. Through a recent (FY20) merger, 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 node 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 considering for standardization 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 multi-component applications such as coupled multi-physics 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

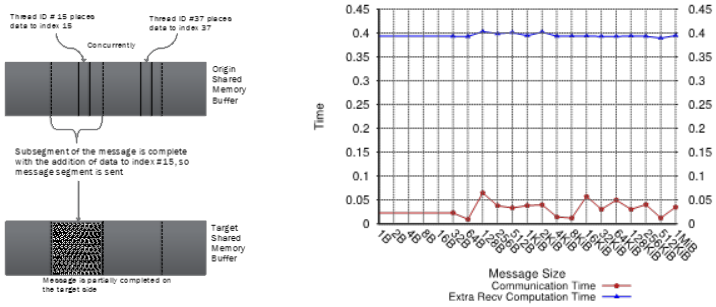


Figure 29: ReInit reduces the time for applications to recover from faults using checkpoints.

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 There has been a great deal of activity within the MPI standards community recently as they work to finalize the feature set for MPI 4.0 by December 2019 and formal adoption by November 2020. A number of error management and recovery features championed by the OMPI-X team have already been formally voted into 4.0. The Sessions proposal, also driven significantly by OMPI-X is considered a mandatory feature for 4.0. And it is considered extremely likely that Partitioned Communications (Figure 29), led by OMPI-X, will also make version 4.0. In the longer-term discussions, OMPI-X is also championing the "ReInit" proposal for simplified checkpoint/restart application recovery (Figure 30) and is actively engaged on other topics relevant to exascale, including MPI_T events, enhanced hardware topologies, thread safety enhancements, half-precision datatypes, and a replacement for the PMPI interface. ECP (mini-) applications which have been used in demonstrating various features and capabilities include: AMG, CoMD, LULESH, miniFE, miniMD, NWChemEx, and SimpleMOC.

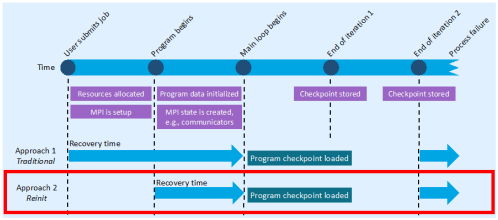


Figure 30: ReInit reduces application recovery time.

Motivated largely by support for the Partitioned Communications proposal and other situations where high network concurrency is required, a general user-level threading abstraction has been developed to support both the Qthreads and BOLT/Argobots threading libraries within either the Open MPI or MPICH libraries. This work, carried out in collaboration with the MPICH and Argobots ECP teams (WBS 2.3.1.07 and

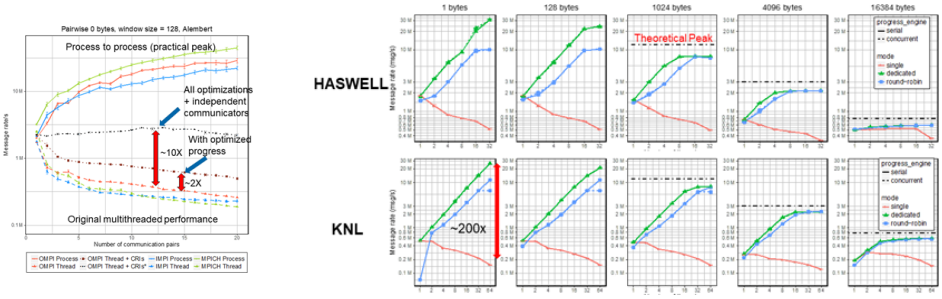


Figure 31: Performance of point-to-point (left) and remote memory access (right) communication with threading.

2.3.5.05) is in the process of being integrated into the two MPI implementations.

To provide better long-term support for resilience, a number of fault detection and coordination features have been refactored out of Open MPI and moved into the PMIx implementation, where they can benefit to a larger community of programming models, including all other runtimes and tools that use PMIx. The OMPI-X team has also played an active role in helping the PMIx community grow and formalize its processes within the last year. This includes separating the standard from the reference implementation, and establishing and documenting governance procedures for both. This will allow the PMIx community to better deal with the strong uptick in interest and actual and potential PMIx clients recently.

Two areas of progress on supporting exacale-relevant architectures include complex memory and topology/congestion awareness within the MPI library. In collaboration with the ECP SICM project (WBS 2.3.1.16), we are integrating support for complex memory architectures into Open MPI. This will enable controlled placement of data objects within different memory spaces according to affinity and/or performance characteristics. The ability to detect and adapt to network topology and traffic patterns is also a valuable capability to facilitate application performance which has been prototyped within Open MPI.

In support of better "MPI+X" interoperability, we have redesigned the internal Open MPI infrastructure to minimize contention on the execution path of independent MPI operations and optimize the handling of MPI requests, drastically improving the communication, point-to-point, RMA and collective, performance of Open MPI in threaded contexts. In addition the progress engine has been reimplemented to maximize the opportunity for computation/communication overlap, and provide a guarantee for asynchronous progress when possible. Using these capabilities we also demonstrated fined-grained control of thread and process placement in the context of OpenMP and Open MPI, in collaboration with the SOLLVE ECP project (WBS 2.3.2.11). Such a capability is crucial to the ability of MPI+X applications to effectively utilize current and future "fat" node architectures (with extensive compute capabilities), but can be hard or impossible to achieve unless the two runtimes are made aware of each other and (minimally) cooperate on resource management. Because poor performance with threading is one of the most common complaints against (any) MPI implementations, a great deal of our recent effort has been devoted to improving this in various ways within Open MPI. We have redesigned point-to-point and remote memory access request management, and redesigned the progress engine for better performance with threads (Figure 31). Moreover, the Partitioned Communications proposal, mentioned above also targets threaded execution by increasing opportunities for fine-grained overlap of computation with communication and reduced locking. We have also improved the message matching implementation so support faster message rates and high thread counts.

Finally, we continue to extend the testing infrastructure for Open MPI and related products, increasing the platforms on which testing is implemented, and expanding the repertoire of tests. We have enhanced the MTT tool used for testing of Open MPI itself, and supplemented it with a new tool aimed at testing of application kernels and performance testing. We have deployed testing capabilities on Summit, and are also testing on Cori.

Next Steps In FY20 and beyond, we plan to continue working across the multiple fronts described above. Finalizing version 4.0 of the MPI standard will be an important part of our work in FY20, as part of the larger community. Since the first exascale systems have now been announced, we can also start incorporating activities targeted to those specific systems into our work as well. Both the Argonne and Oak Ridge systems will utilize Cray's SlingShot network, which is currently under active development. NERSC's Perlmutter system will include SlingShot version 10, while Aurora and Frontier will use version 11.

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 bring 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 Oak Ridge, 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 since RAJA and integrated content were not available at the time of publication.

Overview The Kokkos C++ Performance Portability Ecosystem is a production-level solution for writing modern C++ applications in an hardware-agnostic way. Started by Sandia National Laboratories, 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 Exascale Computing Project, with a mix of NNSA ATDM and Office of Science 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.

Recent Progress The Kokkos project now consists of an integrated developers team spanning five DOE National Laboratories. In particular both NNSA and Office of Science 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 now 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 has begun and initial capabilities for AMD GPUs and Intels GPUs are working. On the AMD side this is despite the fact that full support was previously available, but had to be removed due to the deprecation of the underlying programming model by AMD.

Three Kokkos specific multi-day boot camps were organized with a total attendance on the order of 60 developers. Additionally the team provided single day tutorials at numerous venues with several hundred attendees in aggregate. The slack channel now sees daily questions from numerous users, and has attracted even vendor representatives who help answer machine specific questions. The team finished developing a full API documentation as well as adding use case descriptions for common patterns found in applications.

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. A very recent development, 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 was approved by the relevant study groups, as well as the library evolution incubator group. The Kokkos team was also able to gain co-authors from NVIDIA, Intel and AMD - providing significant support from the leading hardware vendors.

Next Steps The most pressing next task is to complete and then mature support for the upcoming ExaScale architectures. Most of Kokkos's functionality is expected to be available by the end of FY20 for those systems, which will provide another year of time to mature the support before the first platforms will be delivered.

On the programming model evolution side, more explicit asynchronous capabilities including the creation of graphs of kernel launches is a high priority.

At the plumbing level, the team will start to replace some of the implementation level of Kokkos by the ISO C++ capabilities such as atomic_ref and mdspan.

For Kokkos tools an upcoming capability will be the addition of autotuning tools. This will help with the increasingly difficult task of coming up with heuristics to determine good runtime settings for kernels on all the different architectures.

And last but not least, the team will work on getting the proposed new ISO C++ capabilities into the C++23 draft standard, so that they will be available to our users as a vendor provided capability during the lifetime of the first ExaScale platforms.

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

The goal of the Argo project [58] 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:

- AML — a library providing explicit, application-aware memory management for deep memory systems,
- UMap — a user level library incorporating NVRAM into complex memory hierarchy using a high performance `mmap`-like interface.
- PowerStack — power management infrastructure optimizing performance of Exascale applications under power or energy constraints.
- NRM — a daemon centralizing 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 high-performance computing 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 can be used to create, for example, a software-managed scratchpad for multilevel DRAM hierarchy such as HBM and DDR. Such a scratchpad provides applications with a memory region with a predictable high performance for critical data structures.

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.

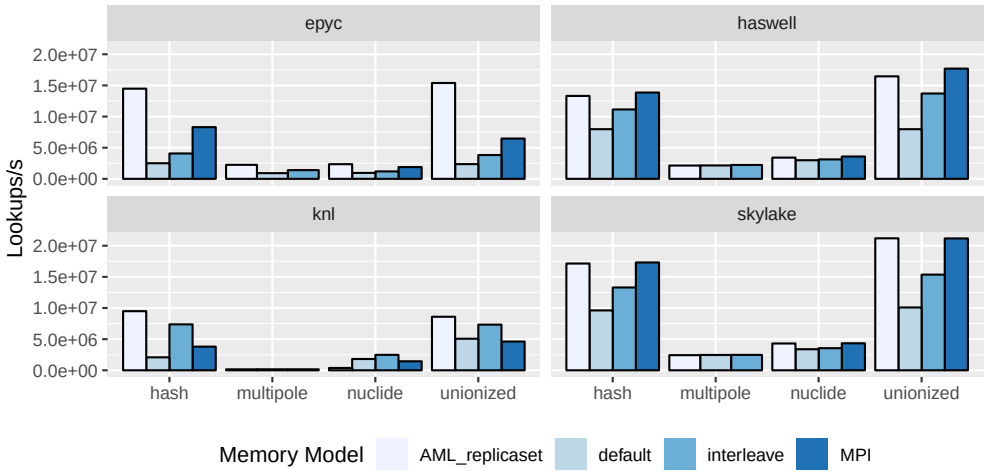
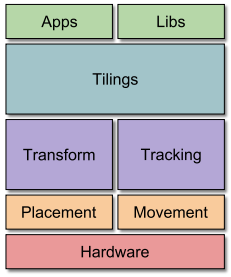


Figure 32: Impact of the memory management policy used on the performance of proxy apps.

The figure on the right depicts the major components of AML, including:



- Topology and hardware management (dependent on NUMA, hwloc, and SICM)
- Data layout descriptions (application-specific)
- Tiling schemes (including ghost areas)
- Data movement facilities (currently primarily `memcpy`)
- Pipelining helpers (scratchpad, asynchronous requests)

Recent Progress AML development reached its first stable release recently. We have a custom CI pipeline in place that ensures this stability via automated testing (we are making progress on leveraging the ECP CI infrastructure as well). The source code of version 0.1.0 is available on our website; we also provide documentation on ReadTheDocs. AML can also be installed via Spack.

We have been collaborating with the ExaSMR project on improving the performance of the XSBench and RSBench proxy apps. We applied the topology-aware data replication facilities of AML in order to replicate read-only and latency-sensitive data on low-latency memory nodes, improving the code's behavior on NUMA architectures. We found the effort to have a limited impact on the application code, provided that the data is correctly structured. Performance was tested on four x86 architectures: Intel's Knights Landing, Skylake, and Haswell, as well as AMD's Epyc. Figure 32 outlines the results of these experiments.

Next Steps We want to add support for more types of memory through the integration with UMap and with other projects such as SICM. We are planning to significantly increase the topology-awareness of our interface through the integration with Hwloc, by providing support for GPUs and other accelerators and their topologies, and by extending the interface with performance-oriented topology queries. We are engaged in Aurora co-design effort to ensure suitable hardware support on that platform. Our positive early results with ExaSMR proxy apps will be expanded to cover the actual OpenMC application.

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 33.

Recent Progress In recent months, we have released UMap v. 2.0, with major enhancements to improve load balancing and incorporate features for additional configurability. We have used UMap in new applications including a key/value store and the lrzip utility.

A paper on UMap will be presented at the Memory Centric HPC Workshop (MCHPC) at SC19.

Next Steps In the coming year we plan to continue outreach to application teams within ECP and in the science/data community. We plan to implement a new UMap handler for the persistent memory Meta Allocator (Metall) which is part of SICM. Metall provides support for persistent heaps as an alternative to data serialization, and is being considered for use by the Exagraph and EXAALT code teams. We plan to add support for disaggregated in-system memory/storage in the core UMap handler to augment the existing file-backed mechanism. This will enable applications to access data on other nodes in the network without requiring explicit remote file system mount points on each node.

PowerStack

Overview Power remains a critical constraint for Exascale. As we design supercomputers 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, they will not be sufficient, 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 while adhering to system, job and node constraints involves complex decision making in software.

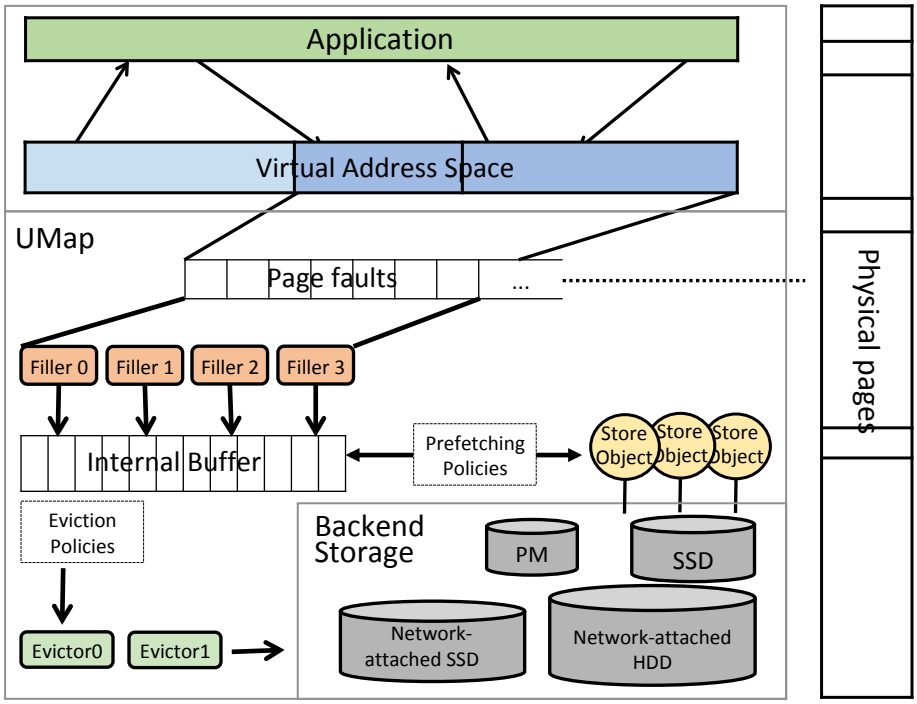


Figure 33: UMap Handler architecture

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. The figure below presents an overview of the envisioned PowerStack, which takes a holistic approach to power management. Additionally, power monitoring and management support for science workflows (such as MuMMI Cancer workflow) and in-situ visualization workflows is 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 Exascale applications to operate safely with optimal performance under power and energy constraints. This project is also essential for building a sophisticated hierarchical software stack proposed by the ECP ATDM (LLNL) and Flux projects. 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 PowerSteering project is to provide a solution for this technology gap.

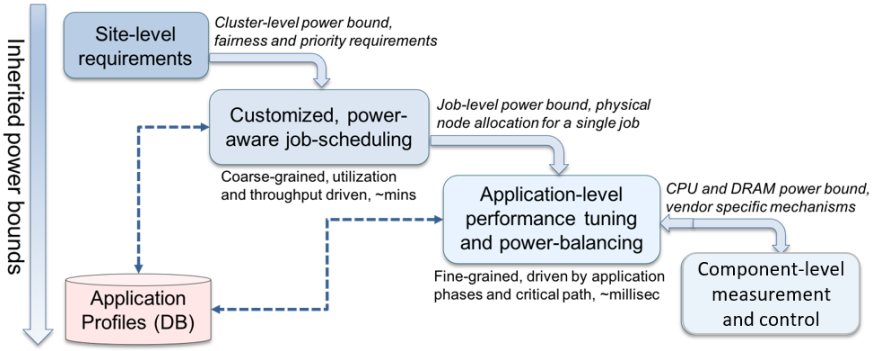


Figure 34: Envisioned PowerStack

Another set of challenges come from portability issues. Existing solutions are targeted toward specific Intel microarchitectures as well as programming models. 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 as well as 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. Such a stack can 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 as well as portability through SPANK plugins. For the *job-level*, a power management runtime system called GEOPM that will optimize performance of Exascale scientific applications transparently under power and/or energy constraints 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, we are closely collaborating with ECP MuMMI workflow project and ECP Flux projects, and with 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 in September 2019. The first was to finish development of GRM (Flux and SLURM Spank plugin), delivered as part of ECP Argo. Here, we developed a SPANK plugin in SLURM to allow for power management across jobs – we chose the SPANK plugin due to its portability benefits and non-dependence on specific SLURM versions. We also released a variation-aware scheduling plugin for Flux, which involved mitigating variation resulting from manufacturing differences under power caps on large-scale clusters.

The other two milestones were delivered as part of PowerSteering project which merged with Argo for FY20. Here, we worked on GEOPM integration on non-intel architecture, and for evaluation of power-aware mappers in the Legion runtime system framework. We chose IBM Power9 (Witherspoon) architecture for GEOPM, which applies well to the Sierra/Summit systems. We developed a DVFS-based model for Intel GEOPM and leveraged GPU NVIDIA's NVML library to create this port. The second milestone involved developing a new power-aware mapper in Legion and creating a two benchmarks for evaluation. This allowed for support of power management tools for non-MPI applications. Releases were made separately for the two milestones on GitHub, and were evaluated on an isolated IBM P9 node (alehouse) and quartz cluster (2K nodes) at LLNL.

We established the PowerStack community charter in June 2018, involving collaborators across multiple vendors (Intel, IBM, ARM, HPE, AMD), 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 1.5 years, this group is looking at engineering and research challenges, along with RFP/procurement designs

through active vendor interaction and involvement. A face-to-face seminar will be held the week prior to SC19 from November 13-15, 2019 in Colorado Springs for the same, the details of which can be found here: https://hpcpowerstack.github.io/powerstack-nov19.html.

Next Steps We will continue our research and development work as planned toward the FY20 milestones. More specifically, we will continue development for variorum library, extend Intel GEOPM's new codebase, 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 also explore co-scheduling challenges in power management (University of Arizona) and development of power monitoring and control tools for science workflows such as MuMMI cancer workflow and in-situ visualization workflows. We will also deploy components across Tri-labs through the TOSS operating system environment,

NRM

Overview Argo Node Resource Manager (NRM) is a daemon running on the compute nodes. It centralizes node management activities such as job management, 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 those 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.

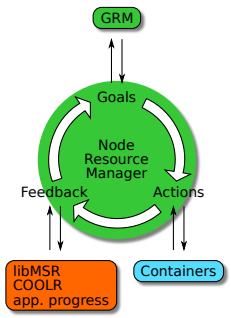
Even in the common case of each compute node being allocated exclusively to a single job, managing available node resources can be a challenge. If a compute node is shared among multiple job components (a likely scenario considering the reduced cost of data transfers), these components—if allowed to 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 *slices* for resource management. Physical resources on the compute nodes are divided into separate partitions. Slices can be used to separate individual components of parallel workloads, in addition to cordoning off system services; this physical separation ensures an improved performance isolation between components.

Our slices can currently manage the CPU cores (hardware threads), the memory and the kernel task scheduling class. The physical resources are partitioned primarily by using the `cgroups` mechanism of the Linux kernel. Work is under way to extend the management to the partitioning of last-level CPU cache using Intel's Cache Allocation Technology.

The NRM daemon maps slices to the node's hardware topology, arbitrating resources between applications and runtime services. Resources can be dynamically reconfigured at run time; interfaces are provided for use from applications and from global services.

The NRM daemon also manages power at the node level. As depicted in the figure on the right, it works in a closed control loop, obtaining goals (power limit) from the global services (GRM), acting on application workloads launched within slices by, for example, adjusting the CPU *p-states* or changing the power cap via the Intel *RAPL* mechanism, and getting feedback through the monitoring of hardware sensors measuring, for example, power draw, temperature, fan speed, and frequency.



We 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.

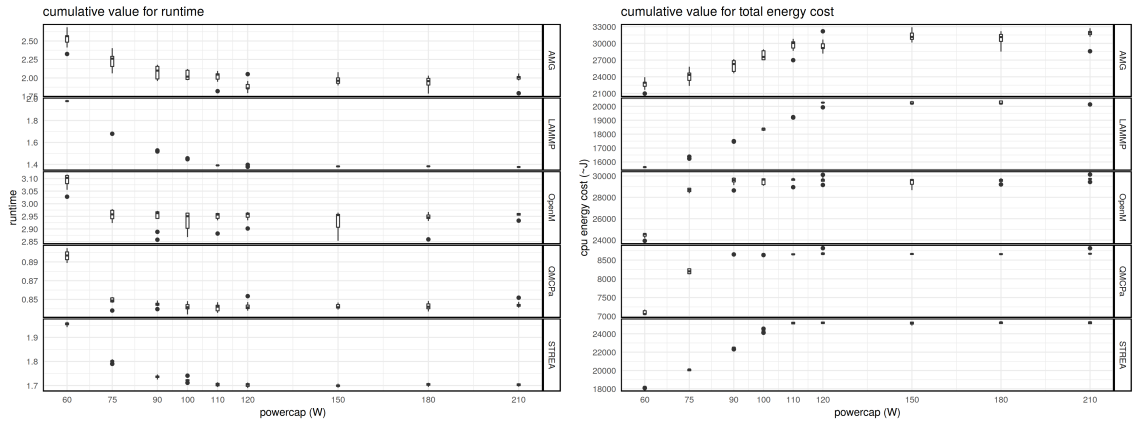


Figure 35: Cumulative run time and energy for different applications and benchmarks running with a varying CPU power cap.

Recent Progress NRM has reached a pre-release quality level, with the source code of version 0.1.0 being available on our website, and the documentation provided on ReadTheDocs. We have a custom CI pipeline in place that ensures the stability via automated testing (we are making progress on leveraging the ECP CI infrastructure as well).

We created the `libnrm` library that can be linked to applications in order to provide reports on application progress to NRM. We made an initial effort of instrumenting a set of ECP application codes and benchmarks: EXAALT, QMCPACK, ExaSMR, AMG, and Stream, with CANDLE underway. This capability gives us insight into the effect of our resource management policies on the run-time behavior of user codes. Among other things, we studied the power/performance tradeoffs of different applications under varying power caps, as depicted in Fig. 35.

Next Steps We are working on expanding the set of ECP applications that are instrumented to report their progress to NRM. The reported data needs to be propagated to the rest of the stack, and a resource management policy needs to be created that takes advantage of this information. We are expanding resource management to comprise of multiple policies, each one optimized for a particular type of workloads, such as BSP, in-situ, and workflows... We are planning to provide a mechanism based on Machine Learning techniques to automatically choose among the available policies. We are also planning to expand interfaces for dynamic node resource control in collaboration with applications, as well as adding an interface for OpenMP codes akin to the PMPI functionality we already have. We want to add support for standardized, OCI-based container runtimes by adding a pass-through mechanism to our resource manager. Finally, we are planning to expand the list of resources managed by NRM by adding support for the partitioning of CPU caches and other vendor-specific mechanisms.

4.2 WBS 2.3.2 DEVELOPMENT TOOLS

End State: A suite of development tools and supporting unified infrastructure aimed at improving developer productivity across increasingly complex architectures, especially those targeted for Exascale platforms.

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. 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 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 development tools built on a unified infrastructure for development, analysis, and optimization of applications, libraries, and infrastructure from the programming environments of the project. 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. 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 seek to establish an open-source compiler activity leveraging activities around the LLVM infrastructure. These efforts will require collaboration and integration with system monitoring and various layers within the software stack.

4.2.4 Plan

It is expected that multiple projects will be supported under the tools effort. To ensure relevance to DOE missions, most of these efforts shall be DOE laboratory led and leverage 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 the Exascale systems. 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 yearly (or more often as needed) based on milestones as well as joint milestone activities shared across the associated software stack activities by AD and HI focus areas.

4.2.5 Risks and Mitigations Strategies

A risk exists in terms of adoption of the various tools and their supporting infrastructure 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 infrastructure 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.

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 [59]. As such, the role of languages, compilers, runtime systems, and performance and debugging tools will becoming 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. We foresee that this trend will continue, which is why we have organized the Development Tools technical area around LLVM-oriented projects. Many of our contributions to LLVM address these trends and will persist 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 Software Technology 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 Hardware Integration 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 presented to the Principle Investigator of each L4 product in May 2019, and the results were tabulated in July 2019. Overall, the products show very good coverage of four of the six areas. In particular, we found that all of the products in the SDK have working Spack packages. This is a very significant finding as it allows us to quickly move ahead to focus on the remaining two areas, testing and GPU support, which need more attention.

In addition to the survey, the Tools SDK was the first ECP project to carry out the integration of an SDK product into the Gitlab Continuous Integration testing workflow proposed by the ECP Hardware and Integration team. Using the GitLab source code repository hosted at the U.S. Department of Energy Office of Scientific and Technical Information and computing resources at the New Mexico Consortium, we successfully demonstrated the essential functionality of the CI workflow. Although there is still much to be done, this represents a substantial milestone in a core component of the sustainability initiative for ECP.

Next Steps Although all of the L4 products in the SDK have Spack packages, only six are in the first release of the E4S distribution. Our first step is to get the remaining Spack packages tested and integrated into E4S to further our KPP3 goal. Additional testing using multiple compilers- including some variant of LLVM currently in use by the Compilers and Debuggers SDK- on at least one current DOE facility machine will be carried out. Results from these tests will continue to be fed back into the L4 products to further guide development of spack packages, bug/issue-reporting workflows, and integration into the greater ECP software

ecosystem. Any discovered issues with Spack, compilers, or libraries will be directly reported back to their respective development teams or L3 representative.

Increasing the number of L4 products in the SDK with CI testing adoption is our second goal. A pilot project using the Dyninst Binary Tools Suite was successfully carried out in FY2019. Using this as a foundation, we propose to include at least an another four of the L4 products into the CI pipeline. Arguably, establishing this workflow is the largest contribution the Tools SDK will bring to the overall ECP software ecosystem. Having automated testing in place across heterogeneous build environments and target architectures is a fundamental challenge to creating reliable, sustainable software- making this work a critical path to attaining the ECP goals of large-scale software sustainability. We also anticipate that this may be the introduction of formal software testing for some of the L4 products. The heterogeneous nature of the testing available in the Tools SDK L4 products will serve as a focused testbed for constructing implementation guidelines for the CI workflow which can then be applied across the SDK efforts and into the greater ECP software ecosystem. Importantly, these lessons can also be carried on by the individual project teams to help maintain their software beyond the ECP timeline.

4.2.8 WBS 2.3.2.06 *Exa-PAPI++*

Overview The Exa-PAPI++ project is developing a new C++ Performance API (PAPI++) software package from the ground up that offers 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. PAPI++ is building upon classic-PAPI functionality and strengthening its path to exascale with a more efficient and flexible software design, one that takes advantage of C++'s object-oriented nature but preserves the low-overhead monitoring of performance counters and adds a vast testing suite.

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 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

1. widening its applicability and providing robust support for exascale hardware resources;
2. supporting finer-grain measurement and control of power, thus offering software developers a basic building block for dynamic application optimization under power constraints;
3. extending PAPI to support software-defined events; and
4. 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 will be channeling the monitoring capabilities of hardware counters, power usage, software-defined events into a robust PAPI++ software package. PAPI++ is meant to be PAPI's replacement—with a more flexible and sustainable software design.

Recent Progress On the **software event** front, the PAPI team has designed and implemented a new API to expose any kind of software-defined events. Since September 2019, the SDE functionality is publicly available through the main repository of PAPI. As a result, software packages that reside in any layer of the software stack can now export information to the outside world in a uniform, well supported, and standardized way. Since the concept of software-defined events is still new to PAPI, the team has worked closely with developers of different ECP libraries and runtimes that serve as natural targets for the adoption of the new SDE API. As of today, we have integrated SDEs into the sparse linear algebra library MAGMA-Sparse (2.3.3.13 CLOVER), the tensor algebra library TAMM (2.2.1.02 NWChemEx), the task-scheduling runtime PaRSEC (2.3.1.09 PaRSEC), and the compiler-based performance analysis tool BYFL (2.4.2 HE).

The examples in Figure 36a illustrate how the convergence of Krylov solvers can be visualized with the help of PAPI SDEs. Each of these solvers behave very differently for different problems and matrices, which, once more, stresses the importance of *exposing these details in a standardized way*. This allows the domain scientist to quickly identify the fastest and most robust method of choice for their very unique problems. Most importantly, this information can now be obtained without expert knowledge about algorithm-specific

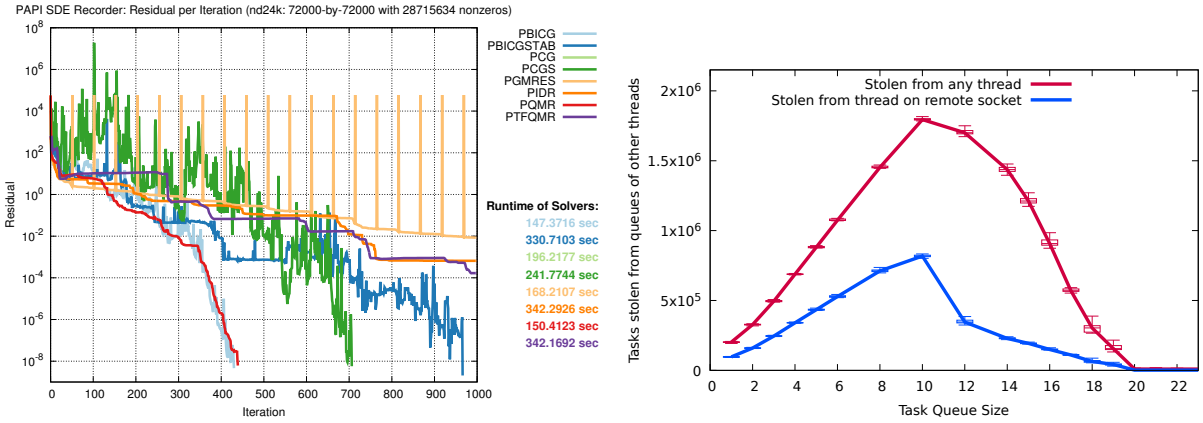


Figure 36: (a) PAPI SDE-Recorders log convergence of different ILU-preconditioned MAGMA-sparse Krylov solvers for a 2D/3D Problem; (b) PAPI SDEs count number of times the scheduler stole tasks from the task queue of another thread in PaRSEC.

characteristics, and without having to instrument MAGMA library code, but simply by calling `PAPI_read()` in the top-level application.

Figure 36b serves as a second showcase, illustrating the evolution of task stealing during the execution of a PaRSEC application that is based on fork-join parallelism with 20 tasks generated at each fork. With SDEs in PARSEC, a user can get a view of what is happening inside the runtime by simply calling `PAPI_start()` and `PAPI_stop()` in their application, without the need to instrument the PaRSEC runtime code.

On the **hardware counter** front, we have developed support for the latest features on NVIDIA Volta GPUs (V100) as featured on the Summit and Sierra systems. Specifically, PAPI users can now monitor both GPU hardware events and the NVLINK performance. Additionally, we developed PAPI capabilities for monitoring power consumption, fan speed, temperature, and power capping support for the V100 GPUs. The latest version of PAPI (5.7.0, released April 2019) has fully integrated support for the NVIDIA GPU counters and power management. Similar efforts are currently in progress enabling users to monitor performance counters as well as power consumption on the AMD Vega GPUs.

Next Steps Our next efforts will focus on:

1. **Formulation of requirements for new PAPI C++ API:** Create and circulate a survey to the ECP teams to assess their needs for hardware and software performance counter functionality. Based on the survey results, we will determine what features are needed for the new PAPI C++ interface. Furthermore, perform a software requirement analysis, and explore novel concepts for expressing software event and hardware counter monitoring through the same PAPI C++ interface.

2. **Decompose PAPI's SDE functionality as standalone library**: The SDE functionality will be decomposed from the PAPI package and made available as a separate library. The production-ready version of the PAPI SDE library will have fully integrated support for enabling SDEs in ECP software layers, as well as monitoring these new events through the PAPI interfaces.

3. **Formulation of a roadmap for refactoring traditional PAPI to PAPI++ software package**: Start the PAPI++ design process for a modular framework that includes a new C++ API in addition to the traditional C and Fortran APIs to preserve backward-compatibility. This effort involves a managed transition away from our legacy PAPI software while continuing to add support for new ECP hardware (released during FY20-21) until the official release of PAPI++.

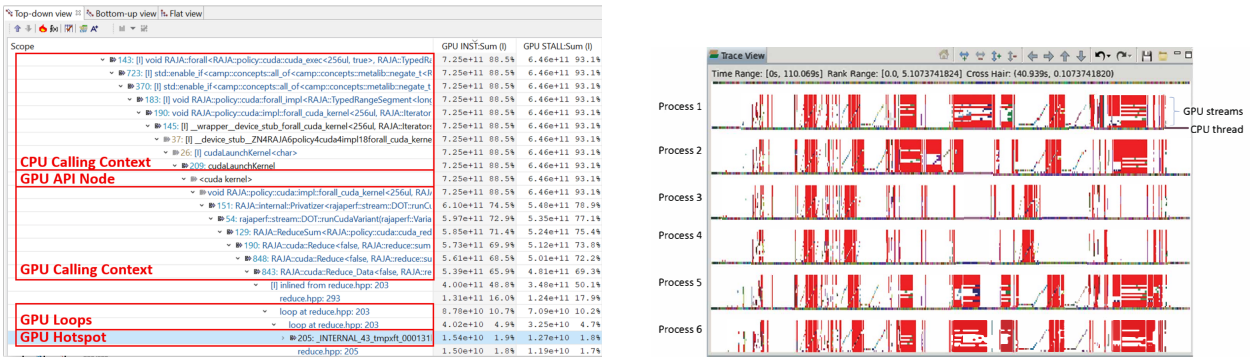
4.2.9 WBS 2.3.2.08 HPCToolkit

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, the next three 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 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 complex node-level programming models used by ECP developers, including OpenMP and template-based programming models 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 37: (a) HPCToolkit's `hpcviewer` showing a detailed attribution of GPU performance metrics in a profile of an optimized, GPU-accelerated benchmark written using LLNL's RAJA template-based programming model. (b) HPCToolkit's `hpctraceviewer` showing a 6-process GPU-accelerated execution trace of Nyx—an adaptive mesh, compressible cosmological hydrodynamics simulation code.

Recent Progress

- The project team developed a novel approach to synthesize a GPU Calling Context Tree (CCT) to report the performance of a GPU kernel that invokes multiple procedures. While attribution of performance metrics in the CCT is approximate, it is essential for understanding the interplay of many procedures that implement a kernel expressed using template-based programming models. Figure 37(a) shows a calling context that spans both CPU and GPU for a benchmark code implemented using LLNL's RAJA template-based programming model.
- The project team extended HPCToolkit with prototype support for and collecting and visualizing traces of both CPU and GPU activity. Figure 37(b) shows a trace of a 6-process GPU-accelerated execution of a recent snapshot of Nyx—an adaptive mesh, compressible cosmological hydrodynamics simulation code being developed as part of the DOE ECP ExaSky project. In the figure, CPU trace lines appear as continuous multi-colored bands running the trace lines the width of the figure. GPU trace lines show large intervals of white, indicating idleness, or red, indicating costly, excess synchronization present in the prototype code measured.
- The project team developed a novel approach to derive key GPU performance metrics, including GPU occupancy, by measuring a single execution of a GPU-accelerated program using program counter sampling on NVIDIA GPUs.
- To reduce the overhead of measuring GPU-accelerated applications, the project team developed a novel wait-free data structure that is employed in the implementation of HPCToolkit's measurement subsystem.

Next Steps

- Integrate GPU measurement and analysis capabilities into HPCToolkit's trunk for release.
- Finish refactoring measurement support initially developed for NVIDIA GPUs into a GPU-independent monitoring substrate.
- Add support for collecting new GPU performance metrics.
- Work with the open source community to upstream GPU measurement support developed by the project team into the community version of the `libomptarget` offloading library.
- 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 seven 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 LLVM (Clacc);
- Use performance modeling and optimization to enable code transformation and performance portability;
- Refine autotuning for OpenMP and OpenACC programming models in order 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; and,
- Aggressively engage applications, SDK, vendor, and software teams to demonstrate and deploy.

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 five topics:

1. OpenACC and Clacc [60]. Develop production-quality, standard-conforming OpenACC compiler and runtime support as an extension of Clang/LLVM. See §4.2.12.

2. Papyrus [61, 62] for portability across NVM architectures. Develop a portable interface to NVM architectures to provide massive, persistent data structures as required by many applications. See §4.2.16.

3. 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 §4.2.15.

4. Improving 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 §4.2.12), and contributing to the development of a new parallel intermediate representation (see `https://github.com/Parallel-IR/llvm-pir/wiki`).

5. Outreach and collaboration with ECP applications teams. We have interacted with over a dozen applications teams to help prepare their applications for ECP. See §4.2.12, §4.2.16, and §4.2.15.

Next Steps: Our next efforts are:

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

2. Papyrus. Improve support for versioning and other performance improvements.

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

4. LLVM Parallel IR. Develop a conceptual prototype for mapping LLVM Clang operations to the proposed Parallel IR, and implement a prototype.

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 toolchain-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 high-performance computing. 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 toolchains 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:

- 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.
- 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 the T-Region mechanism which uses optimizer-understandable runtime calls to represent parallelism constructs. The T-Region mechanism is being implemented upstream, 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.

Recent Progress For parallelism, we have implemented several new features in LLVM:

- A new inter-procedural-analysis framework, and associated transformations, called the Attributor (see [63] for parallelism optimizations). Most of this core infrastructure is now part of the upstream LLVM implementation.
- A new parallel representations, known as T-Regions, and an associated set of parallelism [64].

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

For loop optimizations, we have implemented several new features in LLVM and Clang, and have describe these enhancements in papers ([65, 66] and in several forums directly to the LLVM community (including talks at the LLVM developers' meetings, on the LLVM mailing lists)).

We have developed a prototype C++ compiler, based on Clang, supporting an extension that enables just-in-time compilation [67]. This work has been the subject of a presentation at CppCon 2019 and a keynote talk at the 2019 LLVM developers' meeting. This work is also the basis of a proposal to the C++ standards committee [68].

Next Steps We will continue to develop and upstream the implementations of the developed technologies.

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 a combination of the JIT, 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.

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 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 [69]. However, GCC has lagged behind commercial compilers, such as PGI's, in providing production-quality support for the latest OpenACC specifications [70]. 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 [60] is a major component of the PROTEAS project. 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 objectives for Clacc are:

1. Develop production-quality, standard-conforming OpenACC compiler and runtime 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

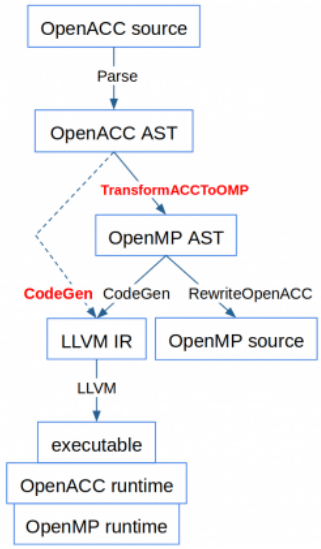
1. **OpenACC Support:** Developing production-quality, standards-conforming OpenACC compiler and runtime support is a large undertaking. Complicating that undertaking further is the need for optimization strategies that are competitive with existing commercial compilers, such as PGI's, which have been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

2. **Source-to-Source:** Source-to-source translation to OpenMP significantly reduces the effort to implement OpenACC and offers additional capabilities, such OpenACC support for proprietary OpenMP compilers. However, a known issue with Clang is that its AST, the source-level representation, was designed to be immutable. Moreover, the AST represents the source after preprocessor expansions, which harm readability and can prevent compilation with other compilers. Finally, 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.

3. **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.

4. **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 include:
   (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.
2. To handle Clang's immutable AST, Clacc's design includes a TransformACCToOMP component that reuses a Clang feature called TreeTransform, which was originally designed for C++ template specializations.
3. To avoid preprocessor expansions in source-to-source mode, Clacc includes a RewriteOpenACC component that reuses a Clang feature called Rewrite.
4. To utilize LLVM IR analyses and optimizations, we are investigating ongoing efforts toward a parallel LLVM IR. Clacc could use such an IR as a code generation target for OpenACC, either directly or after translation to OpenMP extensions Clacc would introduce to support OpenACC's descriptive features.
5. 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++.
6. To ensure Clacc's successful implementation and eventual acceptance upstream, we continue design discussions with the Clang and LLVM communities via mailing lists and other relevant forums.
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 [71] and validation test suites [70] to ensure correct OpenACC behavior and good performance.

Recent Progress

1. Extended Clacc to support additional OpenACC features, including NVIDIA GPU offloading, host-device data-transfer clauses, and the OpenACC Profiling Interface. Prototyped Clang and LLVM support for required OMPT features, including standard OpenMP 5.0 features and extensions.
2. Revamped Clacc's source-to-source mode to improve usability, and addressed feedback from AMD.
3. Contributed numerous improvements to Clang and LLVM, including OpenMP and LLVM testing infrastructure improvements, and to the OpenACC specification, including clarifications for reductions and various clause combinations.
4. Developed report about the future design of OpenACC analyses and optimizations in Clacc.
5. Discussed Clacc at various ECP, HPC, and LLVM venues.

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 specification.

4.2.13 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 within CHiLL. 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 Pascall). Integration with autotuning attains performance that is close to Roofline performance bound for both manycore CPU and GPU architectures.

The Search using Random Forests (SuRF) search framework is a separate tool in Y-Tune that optimizes the search over an autotuning search space. While SuRF provides support to CHiLL for compiler-directed autotuning, it can also be integrated directly with applications and runtimes to search over application parameters and alternative code variants. SuRF is an asynchronous search framework that consists of sampling a small number of input parameter configurations and progressively fitting a surrogate model over the input-output space until exhausting the user-defined maximum number of evaluations. The framework 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. We implemented both MPI- and scheduler-based master-worker approaches.

Recent Progress We have pursued the following main 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. An initial plan for pragmas in Clang and LLVM metadata has been developed. 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. A summer intern has been identified who will work on the JIT/autotuning explorations.

SuRF Supporting Autotuning Search We focused on testing and hardening SuRF for tuning SuperLU package. We used 6 matrices that come from different DOE applications and ran SuRF in an asynchronous mode with up to 32 nodes. We compared the results from SuRF to those from OpenTuner. On all instances tested, we found that SuRF obtains comparable results but in half the time of OpenTuner. We also observed that SuRF found high quality solutions in short computation time and used the remaining time for neighborhood exploration. Therefore, we implemented early stopping criterion. We also did single node tuning experiments with QMC. Since the current search space of QMCPACK is rather small, we did not evaluate it at scale. Currently, we are working with the QMCPACK developers to expose more parameters. Recently, we developed stopping criterion based on local convergence and expected improvement over time. This allows the search to terminate in shorter computation time. Currently, we are expanding the search for multinode autotuning where each evaluation spans multiple nodes. In the past year, we have also used SuRF to perform autotuning search on pragmas, including loop transformations and OpenMP pragmas. Most recently, we are using SuRF to refine descriptive OpenMP pragmas such as `# pragma omp loop` to derive prescriptive pragmas for CPU and GPU mapping of code.

Large high-performance computing (HPC) clusters and DOE leadership-class supercomputing systems pose a few deployment and portability challenges for SuRF. The key issues stem from the differences in queuing systems, scheduling policies, and scripts needed to run the search in a distributed way. Typically, manager worker is implemented with message-passing interface (e.g., MPI) built into the search application. Although this approach is flexible, it requires SuRF to handle a number of system level issues related to system calls (such as apruns, sruns), Python package dependencies, and the correct MPI software stack.

To that end, we integrated Balsam, a default workflow manager on Theta leadership-class system at Argonne Leadership Computing Facility. `BalsamEvaluator` module was implemented to interface SuRF with Balsam. The `BalsamEvaluator` uses the Python API provided by Balsam to interact with the BalsamJob database. Each BalsamJob corresponds to a single autotuning configuration evaluation and contains information pointing to the task executable and the command-line arguments used to run the configuration with the executable. The `BalsamEvaluator` comprise two dictionaries: `pending_evals`, which maps configurations onto the corresponding BalsamJob IDs, and `evals`, which maps the same configurations to the stored objective value (runtime). As a search proceeds asynchronously, receiving data from `BalsamEvaluator`, these data structures are updated accordingly. The `BalsamEvaluator` takes advantage of the Balsam Django API to filter jobs according to their state (e.g., process return code) and leverages functionality such as monitoring job output, logging error tracebacks, and generating compute node utilization profiles.

We developed an easy-to-use common interface for search space definition for autotuning. GPTune is an autotuning software developed within xSDK4ECP project. The interface allow GPTune and SuRF to share the same search space and problem definition. We developed SPACK specifications for SuRF package installation and made the software open source in github.

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 [72, 73].

Next Steps We will experiment with loop transformation and OpenMP pragmas using the pragma autotuner and derive search spaces for these transformations that match patterns in ECP codes. Our goal is to encode patterns that are commonly used by application programmers to simplify the use of autotuning.

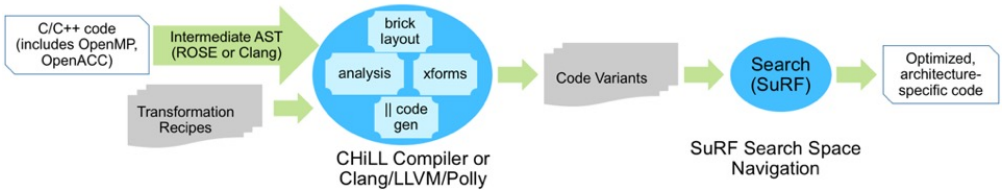


Figure 38: Y-TUNE Solution Approach.

4.2.14 WBS 2.3.2.10 PROTEAS-TUNE - Bricks

Overview We have developed a source-to-source stencil framework ("textttBricks") to address the growing gap between memory and computational performance on pre-exascale systems. The approach uses standard `C++` code to express stencil loops, then transforms the code to use a different memory alignment and ghost zone region to optimize memory and communication performance specifically for that stencil kernel [72, 73, 74]. This also provides an opportunity to inject architecture-specific code transformations, that can take advantage of SIMD/SIMT, threading, virtual memory layouts, and the variety of parameters that are needed to tune for 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 minimal, auto-generated code transformations.

Key Challenges `Bricks` require three primary ingredients for performance portability:
(1) *Stencil kernel metadata* - including stencil radius, dimensionality, neighbor dependencies, and other memory access patterns. For example, for communication the optimal layout depends on the extent of stencil corner coupling and symmetry or reuse.
(2) *Transformation profitability model* - based on the architecture characteristics and benchmarks, determining 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* - knowing what 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 OS or hardware capabilities, like vectorized *shuffle*, or memory *mmap* or *prefetch*, that are required to obtain peak performance.

Solution Strategy With `Bricks`, we have developed a data layout library and code generator for both stencil computations and ghost zone communication. 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 Skylake) and GPUs (NVIDIA Pascal and Volta). Integration with autotuning

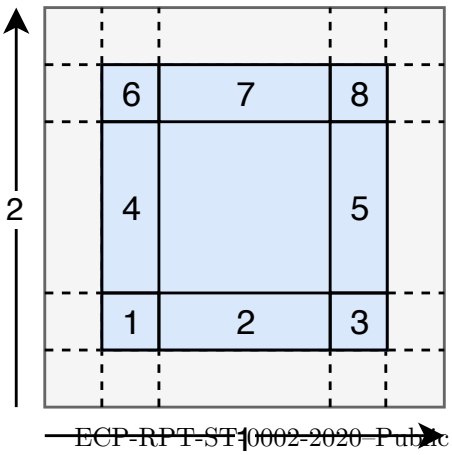


Figure 39: `Bricks` can be used to map memory onto regions that minimize ghost zone packing and MPI message count (2D

attains performance that is close to Roofline performance bound for both manycore CPU and GPU architectures.

Recent Progress For optimization of MPI-based communication on exascale proxy systems, we identified several stencil kernels from applications that leverage the tuned kernels from previous milestones, and evaluated their strong scaling on Theta (Intel KNL) and Summit (NVIDIA V100 GPUs). 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. Because each MPI rank has one or more subdomains with different layouts in memory, this involves "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. This involves both strided memory access and accessing the same array as different messages are sent or received. These operations on the ghost zone "skin" can introduce significant latency and is a blocking operation that, in the limit of strong scaling, can't be hidden by overlapping communication and computation. 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). We have also introduced a novel technique on CPU to significantly reduce the number of messages, using an indirect mapping of memory to MPI buffers.

Next Steps For FY20, we are extending `Bricks` to other application patterns, including block-structured AMR, chemistry kernels, and systems of (non-)linear solvers. 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 Turing), 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.

4.2.15 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. Figure 40 gives an example 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 an 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.

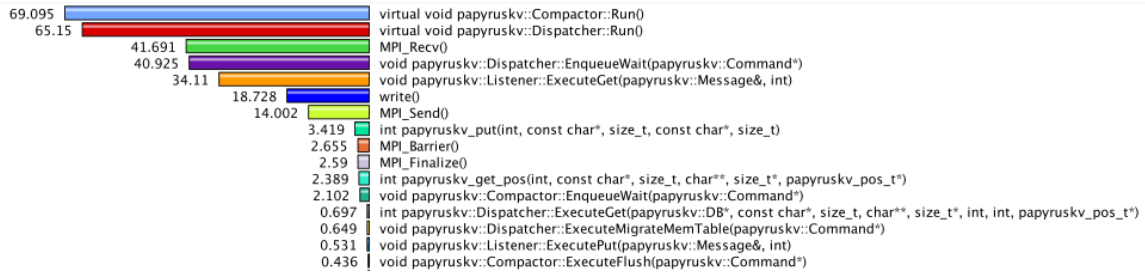


Figure 40: Instrumentation of PapyrusKV library provides insight into asynchronous threaded library details when benchmarking 16 MPI ranks, 10k iterations with 16B keys, 1MB values on Summit using local NVM burst buffers.

Recent Progress

1. **CANDLE** Extended TAU to enhance performance evaluation of multi-threaded Python3 and CUDA and applied it to evaluate the performance of the CANDLE ECP Benchmarks. TAU was extended to provide support for sampling with Python and CUDA.
2. **Improved CUDA and OpenMP support** Updated support for CUDA and OpenCL measurement, organizing data around streams and command queues for robust multithreaded support. Further updated the OpenMP Tools Interface support in TAU, to support the now released 5.0 standard. This OMPT v5.0 standard is currently supported in Intel v19 and Clang v8 compilers.
3. **Flang Instrumentation** TAU support for the Flang Fortran compiler was added. PDT- and Compiler-based instrumentation support for Flang was implemented and tested on x86_64 Linux platforms.
4. **AMReX** TAU OpenACC measurement was demonstrated on the AMReX library.
5. **HIP** TAU's measurement library was extended to support the AMD GPU architecture. Specifically, TAU was extended to support HIP, ROCTracer, and ROCProfiler APIs under ROCm.
6. **CODAR** TAU plugin for streaming profile and trace output to ADIOS2 for realtime application monitoring. Integrated with Chimbuko framework for runtime trace analysis, demonstrated with NWChem on Summit using 2000+ MPI ranks.
7. **NVM Measurement** Instrumented the PapyrusKV library and benchmarked performance of the library while using NVM resources on Summit.

Next Steps

1. **CUDA Enhancements** Implement new Profiling API and Perfworks Metrics API for CUDA/CUPTI 10+ to replace deprecated support for Event API and Metric API.

2. **OpenMP and OpenACC Enhancements** Explore and implement prototype measurement for OpenMP and OpenACC regions executed on target devices.

3. **NVM instrumentation** Design and implement support for supporting deep memory hierarchies in TAU for supporting MCDRAM based systems. Hardware performance counter measurements were enabled using PAPI and LIKWID toolkits.

4. **NVM Measurement** Added profiling and tracing support for NVM architectures.

5. **PHIRE** Improved LLVM IR-based selective instrumentation in TAU at the routine level using PHIRE using a TAU plugin.

6. **Outreach** 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.

7. **E4S** Improved integration of TAU in the E4S.

4.2.16 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) [61], C++ template container library (TCL) [61], and key-value store (KV) [62]. (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.

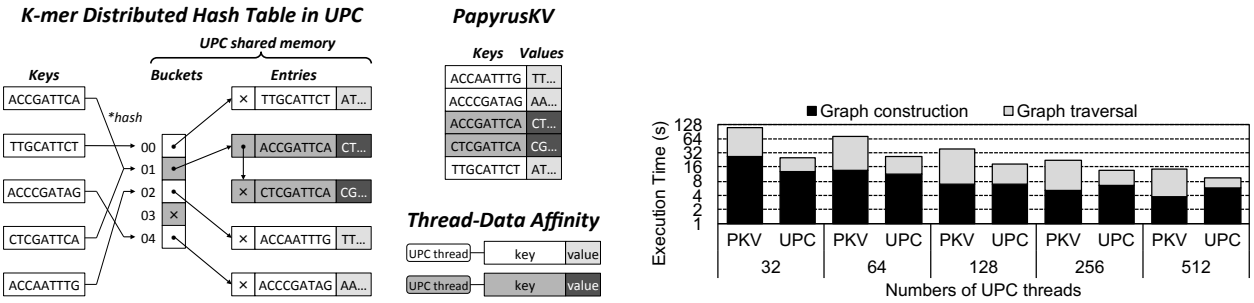


Figure 41: Using PapyrusKV for Meraculous. (a) K-mer distributed hash table implementations in UPC and PapyrusKV. (b) Meraculous performance comparison between PapyrusKV (PKV) and UPC on Cori.

Solution Strategy We describe our key goals for Papyrus: 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 Meraculous [75] is a state-of-the-art de novo assembler written in UPC. Its parallel algorithm for de Bruijn graph construction and traversal leverages the one-sided communication in UPC to facilitate the requisite random access pattern in the global de Bruijn graph. The de Bruijn graph is implemented as a distributed hash table with an overlapping substring of length k, referred to as a *k-mer*, as key and a two-letter code [ACGT][ACGT] as value as shown in Figure 41(a). A hash function is used to define the affinities between UPC threads and hash table entries. We ported the distributed hash table written in UPC to a PapyrusKV database. The keys in the database are k-mers and the values are two-letter codes. The PapyrusKV runtime calls the same hash function in the UPC application to determine the owners of key-value pairs in the database by specifying the custom hash function when the database is created. Thus, the thread-data affinities in UPC and PapyrusKV are the same as shown in Figure 41(a). PapyrusKV requires fewer lines of source code than UPC because it calls standard put and get API functions without implementing an application-specific algorithm for the distributed hash table construction and traversal. Figure 41(b) shows the performance comparison between PapyrusKV and UPC of Meraculous on Cori. Both versions are built and run using Berkeley UPC, an MPI-interoperable UPC implementation. We measured the total execution time on 32, 64, 128, 256, and 512 UPC threads (32 UPC threads per node). UPC shows better performance than PapyrusKV due to its RDMA capability and built-in remote atomic operations during the graph traversal. The performance gap between UPC and PapyrusKV decreases as the number of UPC threads increases. On 512 UPC threads, PapyrusKV runs 1.5 times slower than UPC. This is mainly because of the asynchronous migration in PapyrusKV during the graph construction.

This past year, we have added data compression and encryption to Papyrus. For data compression, the

overhead of data access and movement becomes a serious bottleneck compared to compute overhead in large-scale HPC systems. We integrated data compression methods into Papyrus to achieve storage reduction and performance improvement. For data encryption, we need to protect sensitive data (e.g., health records, DNA data) that is being used in distributed infrastructures, and users need practical methods to secure their data throughout its lifecycle. We will introduce data encryption in Papyrus to add an extra layer of security in the complex scientific workflows.

Next Steps Our next efforts are:

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

4.2.17 SOLLVE

Overview OpenMP is a directive-based API for intra-node programming that is widely used in ECP applications. Implementations of OpenMP and tools to facilitate OpenMP application development.are available in all DOE LCFs. The specification is supported by a stable community of vendors, research labs, and academics who participate in the efforts of the OpenMP Architecture Review Board (ARB) and its Language Committee to evolve its features. The mission of the SOLLVE project is to further enhance OpenMP and its implementations to meet the performance and productivity goals of ECP applications.

SOLLVE has identified open ECP application software requirements, developed features and/or implementation technology to address them, and created use cases that motivate the need for enhancements. The project continues to identify needs and works to standardize them via active participation in the deliberations of the Language Committee.

The project is developing a verification and validation (V&V) suite to assess implementations and enable evaluations by DOE facilities. It is constructing a high-quality, robust OpenMP implementation based on the LLVM compiler. 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 implementers. 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 the challenges by focusing on the following primary activities:

1. **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 5.0 specification. New needs are being identified. This work informs all other project activities by producing use cases, detailed feedback and example codes. It moreover contributes to the OpenMP Examples document.

2. **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. Several proposed features were included in OpenMP 5.0, ratified November 2018. More are under development for version 5.1.

3. **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.

4. **Lightweight OpenMP runtime** The BOLT runtime, built upon ultra-lightweight threading, addresses the need for efficient nested parallelism and improved task scheduling, it develops better support for interoperability with MPI. BOLT is integrated and delivered with the project's LLVM compiler.

5. **Validation and Verification (V&V)** A V&V suite is being implemented 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.

6. **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.

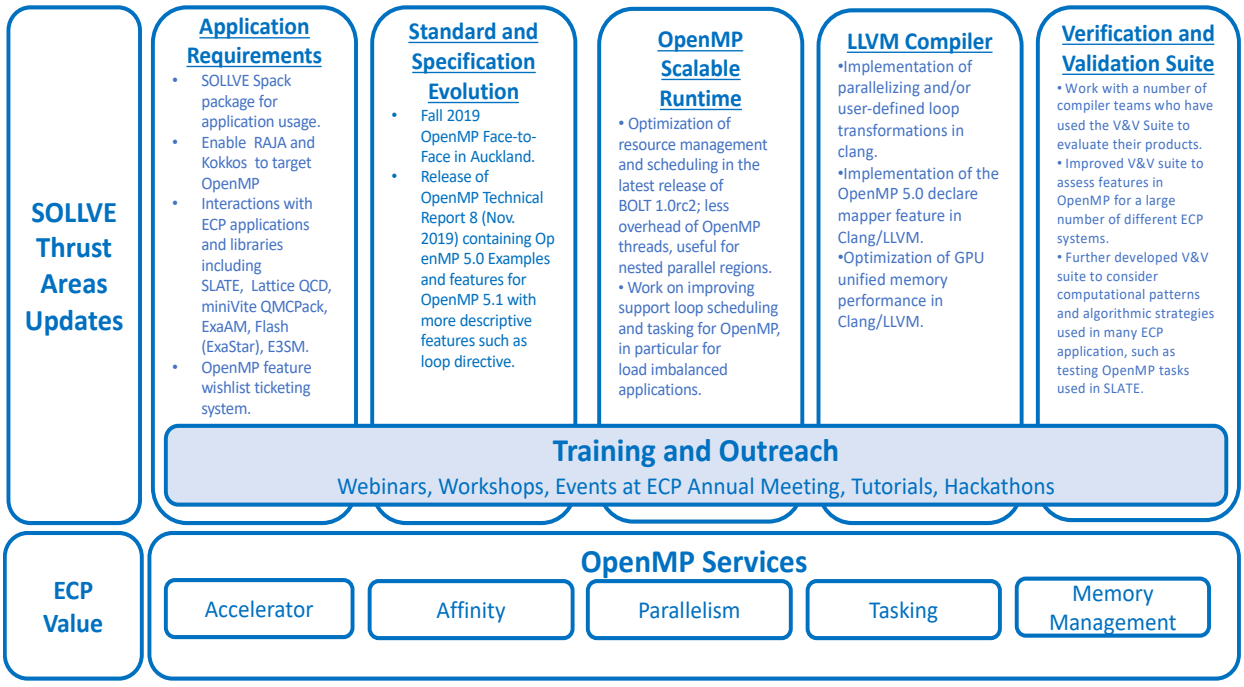


Figure 42: SOLLVE thrust area updates

Recent Progress Figure 42 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 [76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89].

Next Steps The following next steps are planned:

- 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 5.1 is already well under way and is due for release November 2020;
- 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;
- OpenMP runtime: provide support for 5.0 spec; improve performance of MPI+OpenMP codes; address broader set of interoperability challenges; address advanced tasking requirements;
- V&V suite: Continue expanding the coverage of the V&V Suite, with main focus on 4.5 features; expand Fortran tests; work with ARB Examples Committee; improve ALCF toolchains;

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

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 colocated 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 [90, 91, 92, 93, 94, 95, 96, 97, 98] are suitable as common frameworks to facilitate portability and interoperability but offer insufficient flexibility to efficiently capture higher-level abstractions, while specialized runtimes [99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109] 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 [110]. 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 through the following three key aspects:

1. Capturing 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.

2. Exposing hardware computational units through *execution streams* (ESs) as OS-level threads to execute work units. Unlike existing generic runtimes, ESs are exposed to and manageable by users.

3. Allowing 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 43 illustrates the various building blocks in the Argobots framework and the interactions between them to build a hypothetical system.

Recent Progress Integration with other runtime systems is fundamentally important for the Argobots project. BOLT, an LLVM OpenMP runtime over Argobots, is one of the most successful parallel programming systems using Argobots. Thread scheduling mechanism tailored to the OpenMP specification plays a key role in efficient and effective exploitation of nested parallelism in BOLT. As a result, BOLT outperforms existing OS-level thread-based and ULT-based OpenMP runtime systems [3]. This showcases the importance of exposing low-level threading functionalities including customizable thread schedulers and thread pool operations. The Argobots project continues to improve interoperability with communication layers such as MPI runtimes (e.g., MPICH and Open MPI) and Mercury RPC; Argobots benefit these communication libraries by lightweight user-level context switch and flexible user-level scheduling. I/O service is one of the most important application areas for Argobots. Intel DAOS, a high performance storage system developed by Intel, uses Argobots to efficiently handle asynchronous I/O messages. We implemented several features and performed optimizations that are demanded by these key projects, including several debugging functionalities in Argobots.

Overheads of thread creation and scheduling highly impact the performance of applications and runtimes parallelized with Argobots. Our study has proven that a *dynamic promotion technique*, which initially creates ULTs with minimum features and lazily promotes threads when an independent context gets required, are effective in the scenario where chances of suspension are low [111]. In addition to support of x86/64 architectures, we extended this technique to various CPU architectures that are commonly seen in HPC:

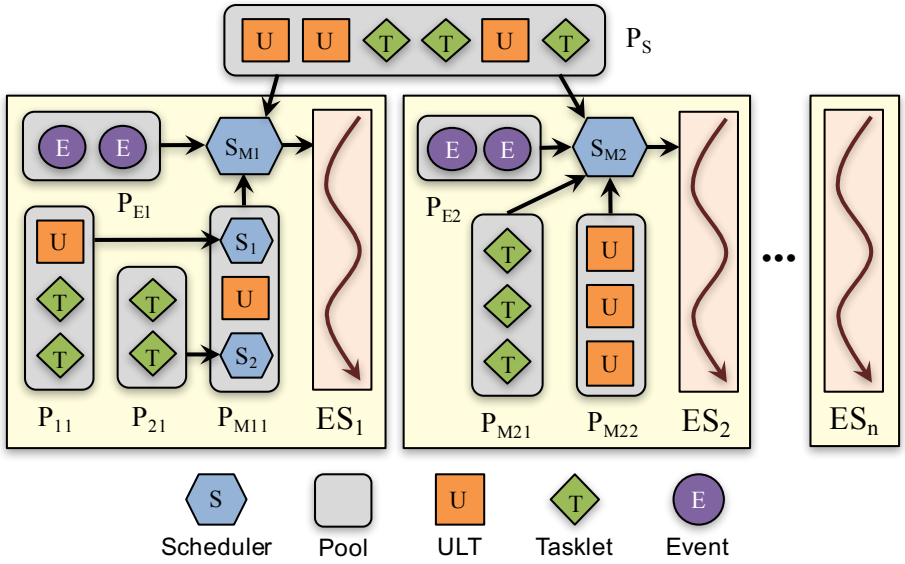


Figure 43: Argobots execution model

POWER and ARM. One of our recent efforts is improvement of testing with continuation integration by increasing test cases, testing compilers, and architectures in order to deliver a stable and well-tested implementation of Argobots.

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. Another key aspect is educating users and growing the user base of Argobots. Our major ongoing and planned steps are as follows.

1. Further integration with communication layers including MPI runtimes and Mercury RPC. Particularly, we continue to collaborate with MPICH and Open MPI developers to make their Argobots interoperability layers more scalable and stable.

2. Enhanced interoperability of multiple Argobots-aware components. As more and more software components use Argobots, managing resources in Argobots across multiple Argobots-aware software stacks gets challenging. For example, blindly creating ESs can incur oversubscription of OS-level threads, which significantly lowers performance. To address this issue, we are investigating an approach that virtualizes the execution stream layer in Argobots.

3. Provides tutorials for users and grows the community. The user community is precious to get feedback and bug reports and stabilize developments as well as expand the Argobots ecosystem. As our tutorial on Argobots at PACT '19 was successful, we plan to have another at PPoPP '20.

4.2.19 WBS 2.3.2.11 BOLT: Lightning Fast OpenMP

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. Our solution is a production quality OpenMP runtime (called BOLT) that leverages 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 [112, 113, 114]. 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.

Key Challenges The growing hardware concurrency in High Performance Computing (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 [111]. Interoperability improvements have also been demonstrated by having BOLT interoperate with an MPI library (MPICH) 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 [110].

Recent Progress Our recent development of BOLT mainly focuses on efficient OpenMP thread scheduling and management [3]. BOLT takes substantial advantage of its base implementation of LLVM OpenMP in terms of ABI compatibility and coverage of functionalities, while we found the necessity of further performance optimizations to fully exploit fine-grained OpenMP parallel regions. After simple replacement of Pthreads call sites with Argobots functions, we adopted several resource management optimizations to reduce contentions and promote reuse of resources. Thanks to low-level scheduling functions exposed by Argobots, BOLT could implement OpenMP affinity tailored to a ULT-based runtime. Our study also the importance of a thread coordination algorithm: it determines how to synchronize with other threads (e.g., busy wait or suspension). To get the optimal performance, existing runtime systems need to tune the wait policy parameter based on the level of oversubscription, while our advanced thread coordination algorithm in BOLT transparently keeps good performance regardless of the degree of thread oversubscription. As shown in Figure 44, BOLT achieves similar performance compared with leading state-of-the-art OpenMP runtimes under flat parallelism, while outperforming all the existing runtimes under nested parallelism.

We note that design and implementation of BOLT are highly regarded in the HPC community. The most significant achievement is the Best Paper Award at PACT '19; our paper on BOLT, titled "BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads" [3] won a Best Paper Award at the 28th international conference on Parallel Architectures and Compilation Techniques (PACT '19), which is considered to be a top tier venue in this field.

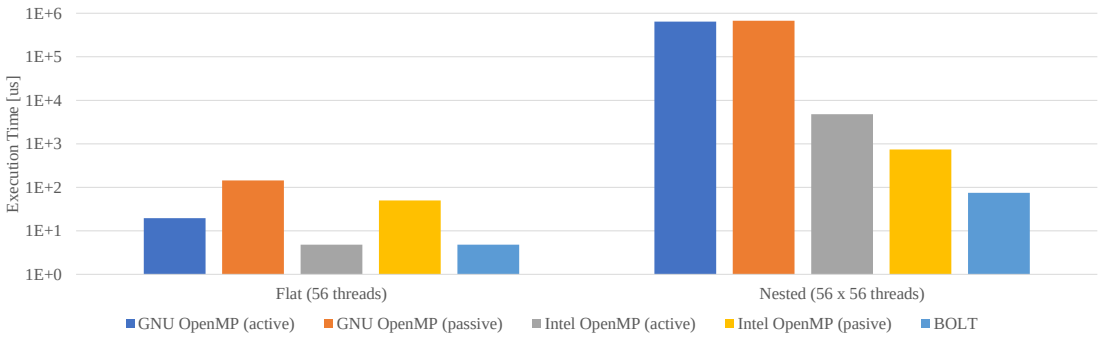


Figure 44: Performance of parallel regions on a two-socket Intel Skylake machine (56 cores in total). GNU OpenMP 8.0 and Intel OpenMP 17.2.174 are used. The flat benchmark creates 56 OpenMP threads while the nested 56 OpenMP threads each of which opens an inner parallel region with 56 threads. We changed the wait policy for GNU and Intel OpenMP. See the paper [3] for details.

The latest BOLT 1.0rc2 has been upgraded to be compatible with LLVM OpenMP 9.0, which further improves performance and functionalities especially for GPU offloading. We also created a development branch that keeps reflecting the latest changes in the LLVM OpenMP's master branch so that BOLT users can try the up-to-date features with ULT support. Interaction and integration are a critical piece for the BOLT project. Our effort in the SOLLVE Spack package, which was one of the SOLLVE milestones, aims at a tighter integration with the other components of the SOLLVE project. Our efforts include interaction and evaluation with (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 (for example, MPICH and Open MPI) 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. Investigates opportunities of utilizing lightweight threads for other optimizations in the context of OpenMP. BOLT successfully enhanced performance of OpenMP threads, while it remains unexplored how BOLT could elevate 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. Improves the interoperability with MPI runtimes. Although the basic scheme is provided via the Argobots layer, its optimization is premature. In addition to improvement of the interoperability between MPI and Argobots, we will investigate an OpenMP-specific interoperability issue if exists.

3. Collaborates with more applications and users. BOLT has gained the attention of the HPC community, but its reach and impact remain limited. In order to find potential room for optimizations and evaluate the performance of BOLT in real workloads, we further investigate other applications, including ECP ones, that BOLT can benefit.

4.2.20 WBS 2.3.2.12 Flang

Overview The Flang project provides an open source Fortran [115] [116] [117] compiler. The project was recently formally accepted as a component of the LLVM Compiler Infrastructure (see http://llvm.org) [118]. Leveraging LLVM, Flang will provide a cross-platform Fortran solution available to ECP and the broad, 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 R&D of interest to a large deployed base of Fortran applications.

LLVM's growing popularity and wide adoption within the broader HPC community make it an integral part of the modern software ecosystem. This project provides the foundation for a Fortran solution that will complement and interoperate with the Clang/LLVM C++ compiler. We aim to allow Fortran to grow into a modernized open source form that is stable, has an active footprint within the LLVM community, and will meet the needs of a broad scientific computing community.

Key Challenges Today there are several commercially-supported Fortran compilers, typically available on only one or a few platforms. None of these are open source. While the GNU gfortran open source compiler is available on a wide variety of platforms, the source base is not modern LLVM-style C++ and the GPL open source license is not compatible with LLVM, both of which can impact 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 robustly supporting all Fortran language features, programming models, and scalability required for effective use on Exascale systems.

Solution Strategy With the adoption of Flang into the LLVM community as an official subproject, our strategy focuses on development and delivery of a solid, alternative Fortran compiler for DOE's Exascale platforms. In addition, we must be good shepherds within the LLVM community to establish and grow a vibrant community of our own. This is in the best interest of ECP as well as the long-term success of Fortran in the LLVM community and the many industry and academic projects that rely upon it.

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 relatively 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 is currently in the process of making a transition from a stand-alone git repository to an LLVM hosted project. This represents a significant result and the current code is available as "F18" via GitHub:

https://github.com/flang-compiler/f18

When it officially completes the move to LLVM's repository it will be renamed "Flang".

The current capabilities of "F18" include the full Fortran 2018 standard and OpenMP 5.X syntax and semantics. As part of the development of the parsing and semantic analysis portions of the front-end, over five million lines of Fortran code has been successfully processed. We will continue and expand this level of testing as we near the completion of the first full (sequential) compiler early in the 2020 calendar year. This effort includes the use of 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.

Next Steps Our short-term priorities are focused on the completion of the sequential compiler, the creation of a significant testing infrastructure, and helping to lead 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 Application Development effort (WBS 2.2) to be responsive to the needs of the applications and (2) collaborate with the other products within the Software Technology 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 Applications Development 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, hypre, Dakota, DTK, 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. The ECP technical directors will schedule quarterly meetings with the ASCR research program managers to get updates on research results that might meet ECP requirements as well as to inform the program managers of ECP needs in applications and software components.

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 six 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 FY20–23, 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, 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 Application Development and Hardware and Integration project focus areas.

4.3.5 Risks and Mitigations 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 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 co-design centers planned in Application Development). 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 Application Development 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.

In addition, 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 short precision can be an order of magnitude faster that double precision.

For a deeper discussion of the futures of ECP Math Libraries efforts, please consult the paper "Preparing Sparse Solvers for Exascale Computing" [119].

4.3.7 WBS 2.3.3.01 xSDK4ECP

Overview The xSDK4ECP project is creating a value-added aggregation of DOE math and scientific libraries through the *xSDK* (Extreme-scale Scientific Software Development Kit) [120], 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 draft 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. Starting FY20, the project will also investigate and deploy multiprecision functionality in the ECP ST ecosystem to enable the use of low-precision hardware function units, reduce the pressure on memory and communication interfaces, and achieve improved performance.

xSDK4ECP 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. xSDK4ECP provides access to all scalable linear solvers with minimal changes. xSDK4ECP is also an essential element of the product release process for ECP ST. xSDK4ECP provides an aggregate build and install capability for all ECP math libraries that supports hierarchical, modular installation of ECP software. Finally, xSDK4ECP 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 such as SuperLU by PETSc and Trilinos. The second, and ongoing phase is increased use of one package's capabilities from another. For example, users who build data objects using PETSc can now access Trilinos solvers without copying to Trilinos data structures. xSDK community package policies [121, 122] 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. xSDK community installation policies [123] help make 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. Regarding the multiprecision effort, the project will assess current status and functionalities, advance the theoretical knowledge on multiprecision algorithms, design prototype implementations and multiprecision interoperability layers, deploy production-ready multiprecision algorithms in the xSDK math libraries, ensure multiprecision cross-library interoperability and integrate multiprecision algorithms into ECP application projects.

Recent Progress Figure 45 illustrates a new *Multiphysics Application C*, built from two complementary applications that can readily employ any libraries in the xSDK, shown in green. Current xSDK member packages (xSDK-0.5.0, released November 2019) are the four founding libraries (hypre [124], PETSc [125], SuperLU [126], and Trilinos [127]); three libraries added in xSDK-0.3.0, released December 2017 (MAGMA [128], MFEM [129], and SUNDIALS [130]); ten more libraries added in xSDK-0.4.0, released December 2018, including seven with DOE support (AMReX [131], DTK [132], Omega_h [133], PLASMA [134], PUMI [135], STRUMPACK [136], and Tasmanian [137]) and three in the broader community (deal.II [138], PHIST [139], and SLEPc [140]); and four new xSDK members (ButterflyPACK [141], Ginkgo [142], and libEnsemble [143], and preCICE [144]). Application domain components are represented in orange. Of particular note is Alquimia [145], a domain-specific interface that support uniform access to multiple biogeochemistry capabilities, including PFLOTRAN [146]. Additional libraries have announced their interest in working toward becoming xSDK member packages and participating in future xSDK releases.

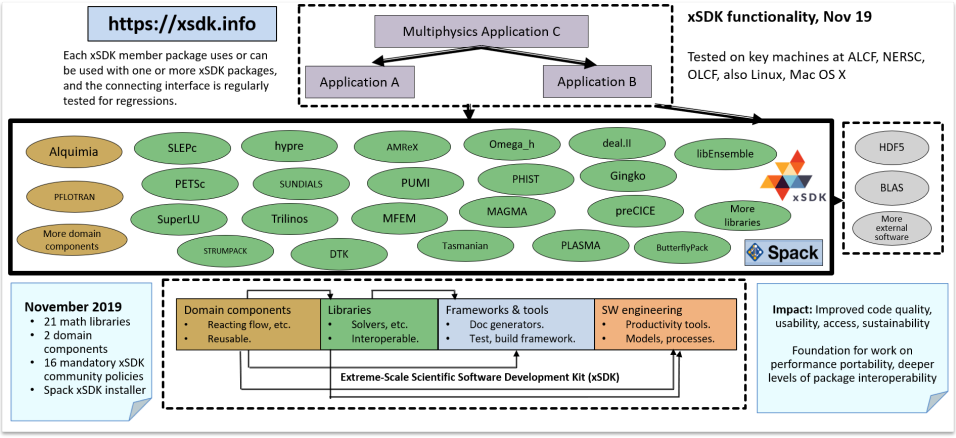


Figure 45: The above diagram shows how multiphysics and multiscale applications can readily access xSDK packages.

The xSDK team also continued development of the community policies[2] with the new 0.5.0 release. The policies were updated, and a new recommended policy was added with feedback from the ECP community. The policies were moved to GitHub, and the process on changing or proposing policies has been updated.

Next Steps Our next efforts include the incorporation of more libraries and the extension of application usage to evaluate the effectiveness of current functionality and to motivate new capabilities. Regarding autotuning of code performance, we will finish a prototype autotuning framework using the multi-output Gaussian process ML approach, which includes multitask and transfer learning. The xSDK4ECP team will also investigate the use of multiprecision for math libraries and document the results of the investigation.

[2]xSDK community policies, https://xsdk.info/policies

4.3.8 WBS 2.3.3.06 PETSc-TAO

Overview Algebraic solvers (generally nonlinear solvers that use sparse linear solvers via Newton's method) and ODE/DAE 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 scales of problem size. Hence, algebraic solver packages provide a wide variety of algorithms and implementations that can be customized for the application and range of problem sizes at hand. PETSc [125, 147] is a widely used software library for the scalable solution of linear, nonlinear, and ODE/DAE systems and computation of adjoints (sometimes called sensitivities) of ODE systems, while TAO provides numerical optimization. We focus on three topics: (1) partially matrix-free scalable solvers 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.

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" simulation (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 described above. This work consist of two parts: (1) the development of libEnsemble and (2) the development of algorithms and software 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.

Running an ensemble of simulation requires a coordination layer that handles load balancing and allows the collection of running simulations to grow and shrink based on feedback. Thus, this 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 are developing 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 using a thread neutral "bypass MPI" approach, which implements default interprocessor communication using MPI but bypasses the use of MPI in performance-critical regions for higher performance and thereby maintains MPI portability.

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 described above, directs the multiple concurrent "function evaluations" needed by optimization, SA, and UQ solvers.

Recent Progress In the past year, we have released PETSc/TAO 3.12 (available at http://www.mcs.anl.gov/petsc), which features enhanced GPU support. Perhaps the most important is the support for CUDA-aware MPI, which allows direct communication of data between Summit GPUs, bypassing the previously needed step of first copying the data to the CPU memory. This enhancement reduces the latency of the communication and improves bandwidth. For example, on Summit for sparse matrix-vector products, this led to a speedup of 33% on one node and a speedup of 13% on on four nodes for the same size problem. Another

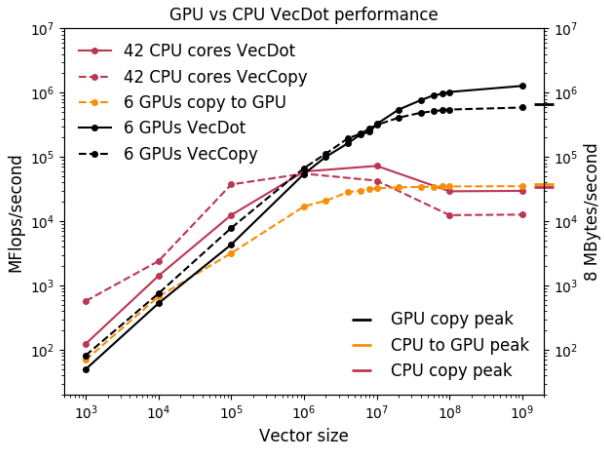


Figure 46: Profile information on the effect of vector size on vector operations compared with memory throughput (one MPI rank per GPU) for PETSc/TAO 3.12. Note the log scale.

important addition is the ability to efficiently read in large meshes on thousands of nodes along with support for collect use of HDF5 calls. With these additions, the algebraic multigrid solver GAMG is up to 12x faster at scale on all of Summit when using the GPUs compared to using just the CPUs.

We have also release libEnsemble 0.5.2 (available at https://github.com/Libensemble/libensemble). Notably, this release includes several changes in progressing toward xSDK compatibility, as well as support for testing on MacOS to support more applications. We also have improved I/O, logging, profiling, and resource detection on Summit.

Next Steps Our next efforts are:

1. **Enhanced application integration and performance optimization and benchmarking**: Refresh the AMReX/PETSc interface and ensure that the PETSc linear solvers can be used with AMReX. Consolidate the two MPI communication modules (VecScatter and PetscSF) to share the same code base and implement two important communication optimizations in the unified code base: internode node-message aggregation optimization, which is critical for scalability of GAMG, and intranode shared memory optimization. Ensure that libEnsemble satisfies xSDK community policies. Continue to produce benchmark results on Summit. Engage with ECP applications and co-design centers to identify new capabilities that can result in FY2021-2023 application integration activities.

2. **Harmonization, performance optimization, and software release**: Harmonize the quasi-Newton and line search modules between SNES and TAO to improve maintainability. Implement GPU optimizations of the quasi-Newton methods. Continue to produce benchmark results on Summit. Continue to engage with ECP applications and co-design centers. Release a new version of PETSc/TAO that maintains compatibility with xSDK community policies.

3. **Enhanced application support and software release**: Implement and optimize the ability to use libCEED under PETSc. Continue to produce benchmark results on Summit. Continue to engage with ECP applications and co-design centers. Release a new version of libEnsemble that maintains compatiblity with xSDK community policies.

4. **Benchmarking, performance optimization and software release**: Demonstrate new libEnsemble functionality by having libEnsemble serve as the driver/outer process that invokes another xSDK tool. Complete benchmarking one or more of the PETSc based applications on Summit at scale including GAMG support on the GPUs and identify performance bottlenecks. Incorporate new desired capabilities resulting from our engagement with ECP applications and co-design centers into our FY2021 plans. Release a new version of PETSc/TAO that maintains compatibility with xSDK community policies.

4.3.9 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 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, Trilinos, 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 multi-GPU 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) develop robust stopping criteria — both absolute and relative — for adaptive (incremental) randomized sampling schemes to reveal numerical ranks in the low-rank compression routine. The goal is to use enough samples for stability, but not too many for efficiency; (2) add OpenMP support for both HSS compression and ULV factorization routines, especially use OpenMP task construct to support irregular parallelism; (3) reduce MPI communication in all stages of the code, including HSS construction, ULV factorization and triangular solve; (4) in addition to HSS, develop codes to support other simpler low-rank formats, such as HOLDR and BLR. The HSS format has asymptotically lower complexity

than HOLDR and BLR, but has a larger prefactor constant. We expect HSS to be more useful for large-scale problems while HOLDR and BLR are more useful for mid-range problems; (5) work with ECP application teams to examine their specific problem characteristics and develop the best clustering/ordering methods to reveal low-rank structures.

Recent Progress In the past six months, we added more support for GPUs and improved on-node threading performance. To this end, we participated in the 3.5-day ECP OpenMP Hackathon at NERSC in August 2019, working closely with the HPCToolkit and SOLLVE teams, as well as the OpenMP/vectorization experts from Intel. Using HPCToolkit revealed several performance bottlenecks. We rewrote the code to remove a few inefficiencies, and identified solution strategies for the other bottlenecks.

We also worked with two ECP applications: MFEM electromagnetic diffusion problems governed by high frequency indefinite Maxwell equations (CEED Co-Design Center) and ExaSGD Miniapp2 – sparse ACOPF matrices: https://github.com/LLNL/hiop/tree/sandbox/matrices/data/acopf_matrices

The algorithmic changes and the results are detailed below.

STRUMPACK

- Algorithmic change: changed recursive tree traversal to a level-by-level traversal, which has less task scheduling overhead and allows to use batched dense linear algebra routines.
- There are many small dense linear algebra operations on lower levels of the supernodal elimination tree. Current solution strategy is to perform small DLA operations on CPU, for larger ones use cuBLAS with CUDA streams. The future solution strategy is to use variable sized batched operations from MAGMA and/or SLATE.
- On Summit, for the MFEM matrix, 8CPU + 1GPU factorization achieved 8x speedup over 8CPU cores, and 19x speedup over 1CPU core.
- Added the new Butterfly and low rank compression schemes. For the MFEM matrix, this obtained much smaller numerical rank and better compression rate compared to the HSS low rank format.

SuperLU

- Developed the first GPU code to perform supernodal sparse triangular solve. On Summit 1 GPU, it achieved 2x speedup over 8 CPU cores for ExaSGD matrix. It is 5.5x faster than Nvidia's cuSPARSE.
- Factorization: on-node OpenMP performance: currently a single "guided" scheduling strategy is used in various OMP for-loops. HPCToolkit points out serious load imbalance for a couple of for-loops. We are examining some hybrid scheduling strategies.

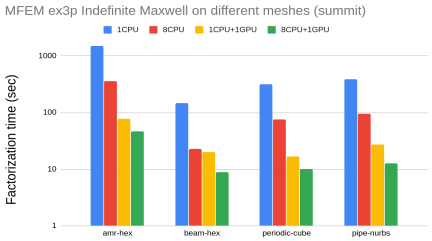


Figure 47: STRUMPACK factorization on Summit GPU.

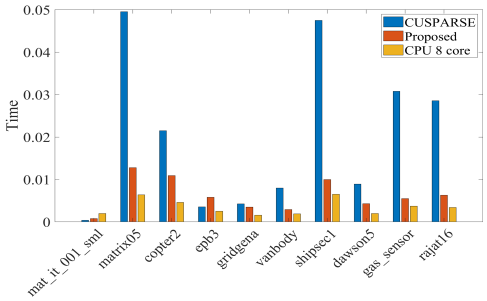


Figure 48: SuperLU solve on Summit GPU.

Next Steps Our future efforts will focus on the following areas: For STRUMPACK, Gather/Scatter operations are still on CPU. The future solution strategy is to use CUDA kernel, or OpenMP 4.5+ target off-load feature. For SuperLU, we will develop a more scalable multi-GPU symbolic factorization code, and develop a communication strategy to combine threading with one-sided communications. We also will build detailed performance models and performance specific code optimizations for the ECP applications that use our solvers.

4.3.10 WBS 2.3.3.07 Sub-project: FFTX

Overview The use of FFTs spans a broad range of DOE science applications, including ones represented in the exascale applications space. Most applications use the API from FFTW, an open-source library developed in the 1990's. FFTW is still used extensively on DOE HPC platforms, and the FFTW API has become the de-facto standard FFT interface: vendors that provide FFT libraries implement (at least a subset) of that interface. Thus, FFTW both defines the standard FFT library interface and is a key software component for applications.

In the FFTX project (https://github.com/spiralgen/fftx), we are developing a new package for supporting FFT applications on Exascale architectures. Our approach is based on two ideas. The first is developing a backward-compatible approach that builds on the FFTW interface but extends it to enable extraction of high performance on exascale machines. The second idea is to provide a toolchain that enables the specialization of the FFT calculation and its surrounding use case calculations (scaling, data layout transformation, marshalling/unmarshalling for communication), using code generation, symbolic analysis, automatic performance tuning, and applications-specific code generation. We will use SPIRAL, an open-source toolchain for FFT developed at CMU, as the basis for developing FFTX, and we will use specific ECP applications and target ECP exascale platforms to provide a focus for this work.

Key Challenges The traditional approach of applications using FFTs is to build up high-performance implementations out of calls to (usually 1D) FFT libraries (either FFTW or vendor libraries), interleaved with user-implemented code for use-case-specific operations. This approach may break down on the emerging ECP platforms, for two reasons. The first is that the node architectures have become more complex. Multiple cores and accelerators lead to multiple levels of parallelism, including threading and SIMD/SIMT. In addition, there are on-node complex memory hierarchies that are to varying extents user-controlled, and across which it is expensive to move data. This leads to more complex interleaving of the other components of multidimensional FFT-based applications with the core library FFTs in order to maximize the effective use of the floating point capabilities and minimize data movement across the memory hierarchy. Some of these are simply not expressible in the current FFTW interface; others can be expressed, but with great effort on the part of the applications programmer, and often with an outcome of not yielding the theoretically-predicted performance due to unexpected and opaque behavior of the FFT library software. A second problem is that the open-source FFTW libraries are no longer supported. The original developers have gone on to other things, and the support of FFTW, such as it is, is performed by volunteer labor. As a result, the extensions to support new node architectures are more brittle and provide less coverage. Expanding the feature set of FFTW to enable the more effective use of the new node architectures is not feasible, since it would entail significant modification and use of the back-end software system, which no one is supporting, and on which the expertise is no longer available.

Solution Strategies There are three components to our approach to providing a new software stack for FFT applications.

(1) We will design an extension of the FFTW interface that both meets the needs of the FFT use cases arising in ECP applications, and exposes the oppportunities for obtaining high performance on current and future architectures. The FFTX interface will be backward compatible with FFTW so that legacy code using FFTW runs unmodified and gains substantially on hardware to which FFTX has been ported. To express the additional opportunities for obtaining improved performance, we will add a small number of new features beyond the FFTW interface to express algorithmic features such as futures/delayed execution, offloading, data placement, callback kernels, and sparsity of inputs or outputs. Such changes will have the potential to extract much higher performance than standard FFTW calls, since higher level operations and new hardware features can be addressed. This interface will be designed as an embedded DSL, for which we will provide a standard C/C++ reference library implementation that enables rapid assessment of the interface by applications developers.

(2) We will develop a new code-generation back end. FFT-based application kernels implemented using the extended FFTW interface described above will be treated as specifications. This approach allows the

extraction of the algorithm semantics from source code and known library semantics, thus providing whole-kernel semantics and whole-kernel specifications. This strategy enables build-time source-to-source translation and advanced performance optimizations, such as cross-library optimization, targeting of accelerators through off-loading, and inlining of user-provided kernels. Our approach also allows for fine control over resource expenditure during optimization. Applications can control compile-time, initialization-time, invocation time optimization resources if needed.

(3) We will develop higher-level FFT-based applications driven primarily by the requirements of ECP applications projects, with the development of interfaces between FFTX and full ECP applications part of the co-design process. The strategy of having a library implementation of the FFTX interface will enable us to use the requirements of ECP applications for the design of the expanded FFTW interface and of the SPIRAL-based toolchain; in addition, the insights provided by opening up the design / tuning space for the constituent FFTs will lead to new ways of designing the applications solvers in order to obtain high performance. We will release the resulting integrated FFT-based packages as a library, called *SpectralPack*.

The core code generation, symbolic analysis, and autotuning software for this project will be based on the open-source SPIRAL software stack, building on 20 years of research by the SPIRAL team at CMU. SPIRAL automatically maps computational kernels across a wide range of computing platforms to highly efficient code, and proves the correctness of the synthesized code. The SPIRAL approach has many of the same structural components as FFTW – a high-level DSL, symbolic analysis, code generation, and autotuning. However, the approach used by SPIRAL integrates these ideas more closely into the user code, generating new source code for both the FFT calls and the glue code (e.g. pointwise operations, data motion) in an FFT-based application.

Recent Progress In the past six months, we implemented FFT use cases relevant to accelerator modeling and materials science using FFTX v1.0. We performed initial code generation for V100 GPU-based systems and baseline performance measurements on Summit. We released reference implementation of FFTX v1.0 for CPUs.

Next Steps Our future efforts will focus on the following areas:

- Deliver high performance on a single node GPU for ECP AD use cases.
- Use "Plan of plans" approach to integrate domain-specific operations with FFTs.
- Design detailed control of data placement / data motion.
- High-level C++ API to improve productivity, broaden adoption.

4.3.11 WBS 2.3.3.12 Sub-project: SUNDIALS

Overview This project is enhancing the SUNDIALS library of numerical software packages for integrating differential systems in time using state-of-the-art time step technologies for use on exascale systems.

The SUNDIALS suite of packages 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. Users can also supply their own nonlinear and linear solvers under the integrators. SUNDIALS is released with data structures supporting several programming environments. Users can employ these supplied structures or provide their own.

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, mainly through incorporation into the discretization-based Co-Design Centers, AMReX and CEED.

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 solution of many small systems of ODEs in parallel on GPUs giving rise to the need for a GPU-enabled ODE integrator that can be used in parallel for many systems at once. Lastly, ECP applications are in need of ODE integrators that can provide increasing functionalities, such as non-identity mass matrices for solving systems of ODEs with finite element spatial discretizations, and maintaining solutions on a constraint manifold.

Solution Strategy This project includes a number of implementation activities that will prepare the SUNDIALS suite of time integrators for exascale systems found in ECP applications. A major activity is the development of support for using the CVODE multistep ODE integration package to evolve multiple systems of ODEs in parallel using GPUs. Two strategies are being pursued for this capability. First, multiple ODE systems are being integrated in each CVODE instance with multiple instances started on a GPU, each with a different CUDA stream. In this case, CVODE is being equipped with a batched direct linear solver capable of using a GPU. In addition, differing methods of choosing time steps and nonlinear solver convergence strategies are being investigated to help optimize performance of CVODE on these aggregated systems. The second strategy is development of a version of CVODE callable on a GPU, although this strategy is most appropriate for very small ODE systems.

This project is also developing efficient ways of handling time-dependent mass matrices for use by the ARKode multistage integrator package and use from within the MFEM library. This capability will likely be supported at the system function evaluation level and will make finite element methods much more efficient with the SUNDIALS multistage integrator. In order to handle ODE systems with constraint manfiolds, an older package, CPODES, is being evaluated and rewritten with the current SUNDIALS interfaces. This constraint manifold capability will be interfaced with ECP application codes for method evaluation and testing.

Lastly, the SUNDIALS team will support ECP applications in interfacing SUNDIALS packages into their software and in the optimal use of the time integration algorithms. This support will also include working with the application teams to help them install SUNDIALS and adjust their build systems to appropriately link with the SUNDIALS library.

Recent Progress In October of 2019, SUNDIALS 5.0.0 was released, including new vector structures that support flexible partitioning of solution data among different processing elements (e.g., CPU + GPU) or for multi-physics problems, an additional vector structure that supports MPI+X paradigms, and interfaces to high performance algebraic solver packages (SUPERLU_DIST, the NVIDIA CUDA-based batched sparse QR direct linear solver, and the PETSc nonlinear solver SNES package). In September of 2019, the SUNDIALS team released a document quantifying performance of SUNDIALS codes on a demonstration problem that solves the three-dimensional nonlinear compressible Euler equations combined with advection and reaction of chemical species run using 40 cores on each of 2 to 3,456 nodes of the ORNL Summit machine. Figure 49 shows a diagram of the many-vector used in the demonstration problem where some solution components are distributed across the MPI processor decomposition and some are fully local. In addition to these activities, the SUNDIALS team has been collaborating closely with the AMReX Co-Design Center team to design effective interfaces to SUNDIALS time integrators from AMReX for applications using ODE integrators, such as for chemistry reaction systems as in Nyx, Castro, and PELE. The PELELM and Nyx applications have both demonstrated use of the CVODE package from SUNDIALS within test runs utilizing new capabilities from SUNDIALS to run on Summit with GPUs.

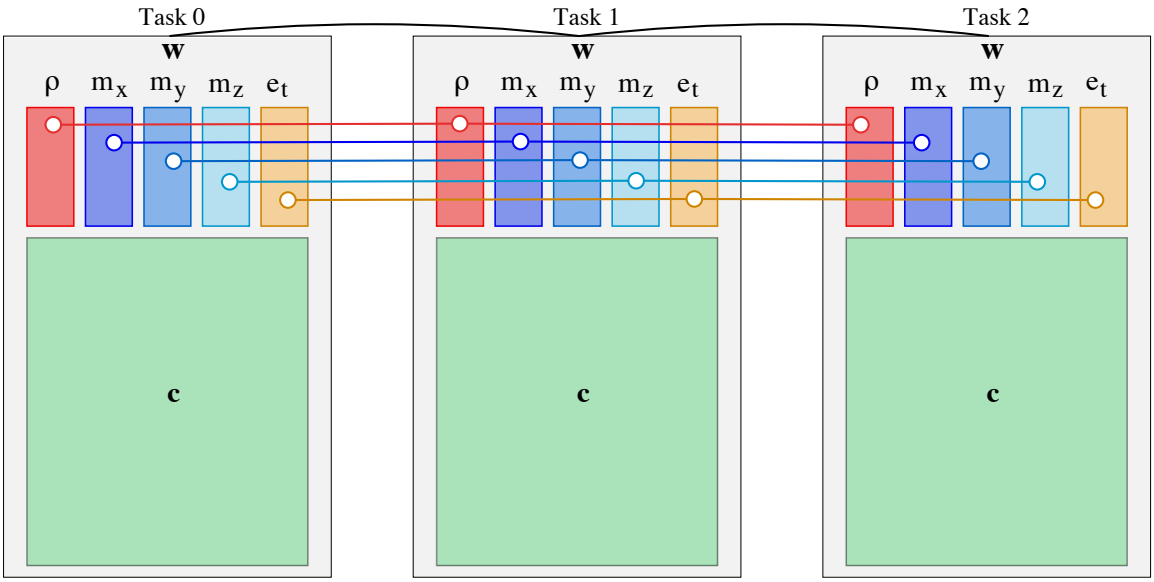


Figure 49: The demonstration code uses the new many-vector capability to store the solution as a collection of distributed (ρ, m_i, e_t) and purely local (c) vectors.

Next Steps During the remainder of FY20, this project team will:

1. Release SUNDIALS with new CVODE options for solving multiple ODE systems in parallel on GPUs.
2. Complete a prototype of CPODES.
3. Release of SUNDIALS with a time-dependent mass matrix capability.
4. Continue to support AMReX and CEED Co-Design Centers in their use of SUNDIALS for ECP applications.

4.3.12 WBS 2.3.3.01 hypre

Overview The *hypre* software library [124, 148] 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 has some 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[149], 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 nonzeroes per row in the resulting matrices and with it increasing numbers of neighbor 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. Current interpolation strategies that lead to best convergence and performance on distributed memory machines are not suitable for implementation on GPUs or similar architectures requiring extreme parallelism. 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 two 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 will also investigate the use of HIP and SYCL for AMD and Intel accelerators that will be available on future exascale computers.

Recent Progress Previously, we enabled the structured solvers, SMG and PFMG[150], both setup and solve phase, to completely run on GPUs, using both CUDA or OpenMP4.5, and to not require unified memory. In addition, options to use RAJA and Kokkos are available. Porting the unstructured solver, BoomerAMG turned out to be far more complex. We first ported the solve phase to GPUs for select smoothers, mainly Jacobi smoothers, requiring unified memory. During the last year, we focused on enabling selected components of the setup phase to run on GPUs, specifically PMIS coarsening, direct interpolation, and the generation of the coarse grid operator, which consists of the multiplication of three sparse matrices. Figure 50 shows runtime comparisons of different implementations of the coarse grid operator generation on one V-100 GPU or one Power 9, which demonstrate that the hypre GPU implementation outperforms state-of-the-art implementations such as CUSP and cuSPARSE. It is now possible to perform a complete linear system solve with BoomerAMG on GPUs using specific settings.

We also implemented a new integer datatype called HYPRE_BigInt to avoid the requirement that all integers need to be converted to 64 bit integers when solving linear systems greater than 2 billions using the unstructured solvers. The new datatype allows to only convert variables that need to be 64 bit integers and improves memory usage and performance, as illustrated in Figure 51.

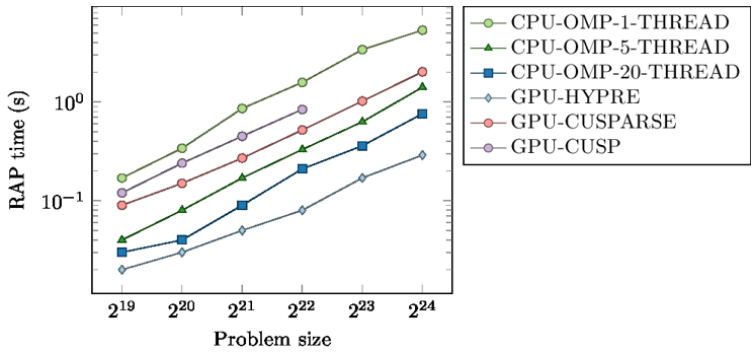


Figure 50: Runtimes to generate a coarse grid operator for a 7pt 3d Laplace problem matrix on 1 V-100 GPU or Power 9 CPU with up to 20 OMP threads for various implementations

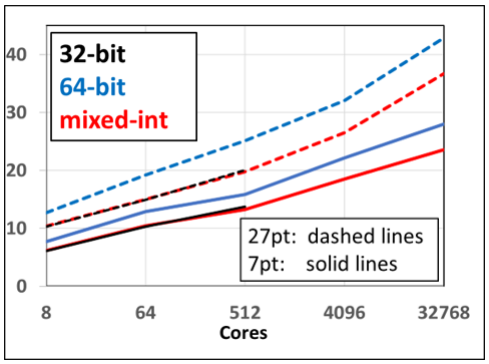


Figure 51: Weak scaling study on LLNL Linux cluster Quartz: Total runtimes in seconds for AMG-PCG using 1M points/core for 2 different 3D diffusion problems.The new mixed-int capability performs about 20-25 percent better than the 64 bit integer version while using less memory and is capable to solve larger problems than the 32 bit integer version.

Next Steps We will pursue the following tasks:

- We will continue to add new GPU capabilities to *hypre*. This includes converting various components that are currently running only on the CPU to be usable on the GPU using CUDA or OpenMP 4.5. We will particularly focus on additional setup components, such as interpolation operators, to achieve better convergence.
- We also plan on improving the efficiency of interfacing applications with *hypre*'s solvers.

In addition, we will work with ECP application teams who are using *hypre*, such as ExaWind, or would like to use it, to achieve best performance by tuning the solvers for them and potentially implementing suitable algorithmic changes.

4.3.13 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 SLATE, FFT-ECP, PEEKS, and Kokkos Kernels 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; FFT-ECP's scope is providing robust and fast calculation for 2D and 3D FFT routines; PEEKS delivers production-ready, latency-tolerant and scalable preconditioned iterative solvers; Kokkos Kernels delivers performance-portable kernels for on-node sparse and dense linear algebra and graph algorithms. Together, these projects form a robust ecosystem of numerical base functionality for Exascale computers.

4.3.14 WBS 2.3.3.13 CLOVER Sub-project FFT-ECP

Overview The FFT-ECP project provides sustainable high-performance multidimensional Fast Fourier Transforms (FFTs) for Exascale platforms. FFT-ECP leverages established but *ad hoc* software tools that have traditionally been part of application codes, but not extracted as independent, supported libraries. These multidimensional FFTs rely on third-party 1D FFTs, either from FFTW or from vendor libraries.

The main objective of the FFT-ECP project is to:

- Collect existing FFT capabilities from ECP application teams;
- Assess gaps, extend, and make available various FFT capabilities as a sustainable math library;
- Explore opportunities to build multidimensional FFT libraries based on vendor 1D FFT kernels and leveraging on-node concurrency from batched FFT formulations;
- Focus on capabilities for Exascale platforms;
- Emphasize leverage of vendor capabilities and the large investment by the broader HPC community in FFT software, and addressing their deficiencies over creation of new and independent software stack.

FFTs are used in many applications such as 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 kernels involved in molecular dynamics (MD) computations, and its performance can affect MD scalability; MD requires solving 3D FFTs of medium size ($10^6 - 10^8$ points). 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.

We found that more than dozen of 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, FFT functionalities will first be delivered to the LAMMPS (molecular dynamics) and HACC (Hardware Accelerated Cosmology Code) ECP applications. LAMMPS and HACC use their own FFTMPI and SWFFT FFT libraries, respectively. FFT-ECP recently provided GPU-acceleration to these libraries. The software release was made under the *Highly Efficient FFTs for Exascale* (**heFFTe**) library [151].

The heFFTe software stack is illustrated in the left-hand side of Figure 52, while the main components of the heFFTe framework are illustrated in the right-hand side of Figure 52. The first and last step address the need for a flexible FFT API to take application-specific input and output (bricks/pencils), including arbitrary initial decompositions. Currently, heFFTe provides efficient GPU support for all communication primitives and features in FFTMPI and SWFFT.

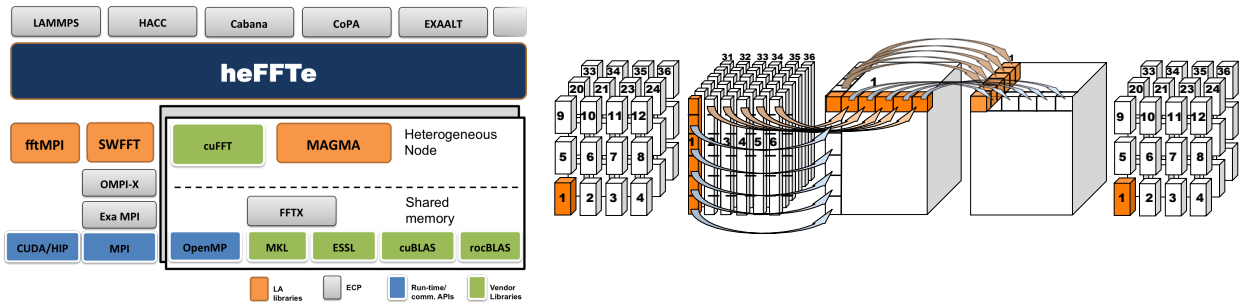


Figure 52: Left: the heFFTe software stack. **Right**: 3D FFT computational pipeline in heFFTe with: 1) Flexible API for application-specific input and output, including bricks/pencils/etc.; 2) Efficient packing/unpacking and MPI communication routines; 3) Efficient 1D/2D FFTs on the node.

Key Challenges

1. **Communication costs:** Communication costs are main bottleneck on current systems; this includes low node bandwidth (relative to high compute capabilities), sub-optimal accelerator-aware MPI communications, and encountered performance degradations in MPI implementations.
2. **Appplication 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.
3. **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

1. **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 [152]. Other strategies include the use of mixed-precision calculations [153, 154] and data compression for reduced communications (including lossy, e.g., using ZFP compression).
2. **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 at a second step to support other ECP applications.
3. **Autotuning:** Performance portability will be addressed through use of standards (like 1D FFTs from vendors), portable linear algebra (LA) using MAGMA [155], and parameterized versions that will be tuned across architectures. We have extensive expertize and well proven track record in the development and use of autotuning techniques for important LA kernels [156, 157].

Recent Progress The FFT-ECP team completed a phase of implementating optimizations and features [151] and released the heFFTe v0.1 library [158]. heFFTe demonstrates very good strong scalability and performance that is close to 90% of the roofline peak (see Figure 53) [158, 152].

Next Steps Next steps of work are using heFFTe in applications, generalizing the APIs to fit application use, adding new APIs as needed, developing a benchmarking framework for MPI FFT communications, and implementing MPI-based optimizations for Summit. Another aspect of work is the development of HIP heFFTe to support AMD GPUs.

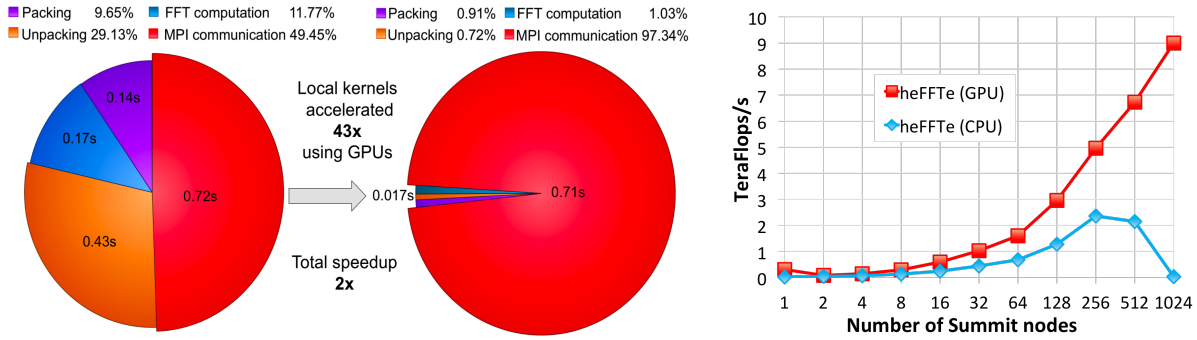


Figure 53: Left: heFFTe acceleration of 1024^3 FFT on 4 Summit nodes. Note: nodal computations are accelerated 43×. **Right**: heFFTe strong scalability on 1024^3 FFT on up to 1024 nodes (×6 V100 GPUs; double complex arithmetic; starting and ending with bricks; performance assumes $5N^3 log_2 N^3$ flops).

4.3.15 WBS 2.3.3.13 CLOVER Sub-project Kokkos Kernels

Overview The Kokkos Kernels [3] 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, 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 timeframe 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 kernel 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.

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 Our primary solution strategy to address these challenges are:

1. **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

[3] https://github.com/kokkos/kokkos-kernels

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.

2. **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.

3. **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 precondtioners, to reduce the number of kernel launches.

Recent Progress

1. Kokkos Kernels has developed team-level linear algebra kernels for several BLAS and LAPACK operations so that ECP applications can use these foundational operations within their device level code. This key design championed by the Kokkos Kernels team is becoming more common place with several vendors adopting such a design. This design reduces synchronization overhead and encourages reuse of the data in the memory hierarchy between several BLAS or LAPACK operations with team level synchronization. This has resulted in better performance in applications like SPARC CFD simulation.

2. Kokkos Kernels preconditioners such as Symmetric Gauss Seidel precondtioners are integrated into the Exawind application. The multicoloring based Symmetric Gauss Seidel precondtioner has resulted in up to 4.5x improvement in overall solve time of the SGS solver.

3. Kokkos Kernels BLAS and sparse linear algebra kernels were integrated into the spectral partitioner of the Exagraph project. Using Kokkos Kernels results in faster spectral partitioning than vendor provided implmentations. This was the result of careful tuning of the kernels for Exagraph needs.

4. Kokkos Kernels team has developed and integrated tutorial materials to the Kokkos tutorials. The tutorials are maintained as a common resource for the entire Kokkos ecosystem.

Next Steps Kokkos Kernels team is focused on:

1. **A major software release**: Kokkos ecosystem 3.0 release will be available to the ECP applications in FY 2020. This includes several new kernels that are requested by ECP applications, performance improvements of kernels that are already being used by ECP applications, support for new architectures such as ARM based systems, and several software changes such as support of standalone CMake.

2. **Developing new kernels to reduce synchronization costs**: Kokkos Kernels team is working on kernels that redude the number of kernels launches by focusing on block-based approaches. This will allow further performance in ECP applications such as Exawind and EMPIRE.

3. **Collaboration with vendors**: Kokkos Kernels team is working with vendor libraries team to incorporate ECP application needs in the vendor library roadmap. Several changes from vendors are expected in FY20. These changes will be added to Kokkos Kernels so ECP applications can get access to the improvements.

4.3.16 WBS 2.3.3.13 CLOVER Sub-project PEEKS

Overview The PEEKS subproject is a focused team effort to advance the capabilities of the ECP software stack in terms of communication-avoiding Krylov solvers and advanced preconditioning techniques featuring fine-grained parallelism. Previously developed techniques that are available as prototype codes – as well as novel algorithm developments – are turned into production-quality implementations and integrated into the ECP software ecosystem as part of the Trilinos [4] and the Ginkgo [5] software stacks.

Key Challenges Developing preconditioned iterative solvers for the US flagship supercomputers deployed in ECP, we acknowledge three major challenges coming from the hardware architecture:

1. Fine-grained parallelism in a single node that has to be exploited efficiently by the iterative solver and the preconditioner.
2. Rising communication and synchronization cost.
3. Computational power growing much faster than memory power, resulting on increased pressure on the bandwidth of all cache/memory levels.
4. Low-precision special function units like Tensor cores that are increasingly adopted by hardware architectures require sophisticated numerical schemes to be useful for general purpose scientific computing.

All 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 adopting multiprecision algorithms for efficient hardware utilization. In the last few decades, numerous efforts have investigated the potential of communication-avoiding (CA) and pipelined Krylov solvers [159, 160], as well as new preconditioning techniques that allow for the efficient parallelization of the preconditioner setup and the preconditioner application [161, 162, 163]. However, most implementations were experimental and rarely adopted by application code. Also the concept of accelerating iterative methods by using lower precision formats for parts of the computations or memory access was extensively investigated in literature [164, 165, 166], while production-ready implementations are still scarce.

Solution Strategy The primary thrusts of the PEEKS project are:

1. **Architecture-portable software design:** In the Ginkgo C++ software, we design and develop a next-generation sparse linear algebra library able to run on multi- and manycore architectures. The library design decouples algorithm implementations from hardware-specific kernel implementations, thereby allowing extensibility as well as architecture-specific kernel optimization.
2. **Sustainability efforts:** The Ginkgo software development cycle adheres the Better Scientific Software (BSSw) design principles [167] 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 [168]. Ginkgo is an open source effort licensed under BSD 3-clause and ships with the latest version of the xSDK package (v.0.5.0).
3. **Pipelined and CA Krylov methods:** We realize 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 [169].
4. **ParILUT – A parallel threshold ILU:** We are spearheading the manycore-parallel computation of threshold-based incomplete factorization preconditioners [170, 171].
5. **Adaptive precision block-Jacobi:** We realized a production-ready block-Jacobi preconditioner that reduces the runtime by carefully selecting the storage format of the distinct block inverses without impacting the preconditioner quality [172].

[4] https://trilinos.org/
[5] https://github.com/ginkgo-project/ginkgo

6. **Software-defined events (SDE):** We team up with the ECP Exa-PAPI project to design and realize an ecosystem for software-defined events. The idea is to provide easy access to library-, domain- and solver-specific metrics via the PAPI interface. This avoids cumbersome code instrumentation and library recompilation for debugging algorithm behavior or identifying performance bottlenecks [173].

Recent Progress

1. For improving the Ginkgo software quality and performance reproducibility, we realized a continuous benchmarking system permanently evaluating the performance of central building blocks and archiving the data [168]. We also realized a web-based Ginkgo Performance Explorer that allows interactively exploration of archived performance data [174].
2. We implemented and released five variations of communication-avoiding and pipelined Krylov solvers in the Belos Trilinos package.
3. We demonstrated the efficient use of communication-avoiding Krylov methods in Trilinos inside wind turbine simulations of the ECP ExaWind project [169].
4. We deployed ParILUT, the first production-ready manycore-parallel algorithm for generating threshold-based incomplete factorization preconditioner and demonstrated significant speedups over state-of-the-art algorithms [171] (see Figure 54).

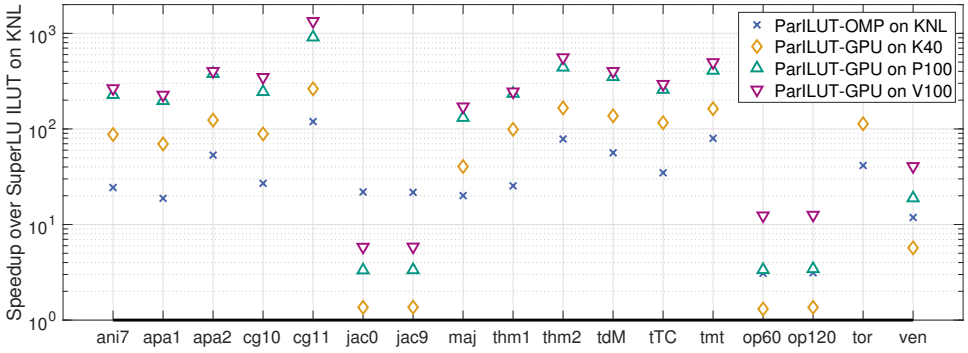


Figure 54: Speedup of the ParILUT over conventional threshold-ILU generation on different manycore architectures. Test problems are taken from the Suite Sparse Matrix Collection.

Next Steps Our next efforts are:

1. **Low-synchronous orthogonalization:** The success of communication-avoiding Krylov methods motivates to push the synchronization limits further by deploying low-synchronous orthogonalization methods. (Collaboration with the ExaWind team at NREL.)
2. **Parallel incomplete factorization preconditioner application:** With the advances in the parallel incomplete factorization preconditioner generation, the focus increasingly turns to the efficient preconditioner application. We enhance the concept of sparse approximate inverse approximation for incomplete factorization preconditioners, and extend the scope to novel hardware architectures featuring attractive performance in the low-precision regimes.
3. **Multiprecision sparse matrix formats:** Operations with sparse matrices are memory-bound on virtually all architectures. We investigate how splitting the matrix int several operators stored in value-optimized less complex floating point precision formats can help improving performance.
4. **Polynomial preconditioners:** The communication cost of numerical preconditions is high. In particular for communication-avoiding pipelined Krylov methods, the synchronization necessary by standard preconditioning can become a bottleneck. We will deliver a new polynomial preconditioner in Trilinos (Belos) and investigate their effectiveness for ECP applications.

4.3.17 WBS 2.3.3.13 CLOVER Sub-project SLATE

Overview The Software for Linear Algebra Targeting Exascale (SLATE) provides fundamental dense linear algebra capabilities to DOE and the HPC community at large. To this end, SLATE provides parallel Basic Linear Algebra Subprograms (BLAS), norms, linear systems solvers, least square solvers, singular value and eigenvalue solvers.

The ultimate objective of SLATE is to replace the venerable Scalable Linear Algebra PACKage (ScaLAPACK) library, which has become the industry standard for dense linear algebra operations in distributed-memory environments. After two decades of operation, ScaLAPACK is past the end of its life cycle and overdue for a replacement, as it can hardly be retrofitted to support GPUs, which are an integral part of today's HPC hardware infrastructure.

Primarily, SLATE aims to extract the full performance potential and maximum scalability from modern HPC machines with large numbers nodes, large numbers of cores per node, and multiple GPUs per node. For typical dense linear algebra workloads, this means getting close to the theoretical roofline peak performance and scaling to the full size of the machine. This is accomplished in a portable manner by relying on standards like MPI and OpenMP.

SLATE functionalities will first be delivered to the ECP applications that most urgently require SLATE capabilities (NWChem, GAMESS, EXAALT, QMCPACK, CANDLE, etc.) and to other software libraries that rely on underlying dense linear algebra services (STRUMPACK, SuperLU, etc.). Figure 55 shows the role of SLATE in the ECP software stack.

While the initial objective of SLATE is to serve as a successful, drop-in replacement for ScaLAPACK with support for GPU accelerators, the ultimate goal of SLATE is to deliver dense linear algebra capabilities beyond the capabilities of ScaLAPACK. This includes new features such as communication-avoiding algorithms and randomization algorithms, as well as the potential to support variable size tiles and block low-rank compressed tiles.

Figure 55: SLATE in the ECP software stack.

Key Challenges

1. **Designing from the ground up:** The SLATE project's primary challenge stems from the need to design the package from the ground up, as no existing software package offers a viable path forward for efficient support of GPUs in a distributed-memory environment.
2. **Facing harsh hardware realities:** SLATE is being developed in a difficult hardware environment, where virtually all the processing power is on the GPU side. Achieving efficiency requires aggressive offload to GPU accelerators and careful optimization of multiple bottlenecks, including interconnect technology lagging behind the computing capabilities of the GPUs.
3. **Facing harsh software realities:** SLATE is being developed using cutting-edge software technologies, and relies on modern C++ features and recent extensions to the OpenMP standard, many of which are not fully supported by compilers and their runtime environments. In terms of GPU acceleration, standardized solutions are still in flux. Also, while complete parallel programming frameworks exist, at this stage they have to be considered research prototypes.

Solution Strategy

1. **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 [175], and followed with a phase of laying out the initial design [176]. Since then, the team rolls out new computational routines every quarter. While we continue to refactor as needed to achieve high performance, the basic design has solidified and been published [177].

2. **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 in the case of computing matrix norms. Care is taken to hide communication by overlapping it with GPU computations.

3. **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 Cray, IBM, Intel, NVIDIA, AMD, and ARM.

Recent Progress During 2019, the SLATE team continued to add to its suite: mixed-precision linear solvers for LU and Cholesky in March, matrix inversion for LU and Cholesky in June, and Hermitian eigenvalue and singular value solvers in September. Routines are available in the four standard precisions (single, double, complex-single, complex-double). In addition to SLATE's native C++ API, compatibility APIs for LAPACK and ScaLAPACK users are also provided, so that SLATE can serve as a drop-in replacement. All developments are documented in SLATE Working Notes [6]

Next Steps

1. **Users' Guide and Developers' Guide:** These guides will supplement SLATE's online Doxygen documentation by thoroughly explaining both SLATE's public interface—how to integrate it into an application, with examples—as well as SLATE's internal design, to help new developers and outside collaborators understand the codebase. As living documents, they will be continually updated as SLATE matures.

2. **Band solvers:** To SLATE's existing general band LU solvers, we are adding Hermitian band Cholesky factorization and solve, along with Hermitian band BLAS and norm routines.

3. **Performance improvements:** We have observed several routines that are not performing as well as expected. In some cases, such as in QR, LQ, and parallel norms, we have already identified improvements to be made to the algorithm, such as refactoring loops to improve parallelism. In other cases, we are analyzing traces to identify and correct problems.

4. **Generalized Hermitian eigenvalue solver:** We will extend SLATE's Hermitian eigenvalue solver to handle generalized Hermitian-definite eigenvalue problems, of the form $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$. These are of strong interest in mechanics and chemistry applications, among others.

5. **New C++, C, and Fortran application programming interfaces (APIs):** The BLAS and (Sca)LAPACK APIs have served well over the past 40 years, but are rooted in out-dated FORTRAN 77 limitations such as cryptic 5–6 letter abbreviations (e.g., `dgemm, dgesv`). For SLATE, we will design a new, user-friendly C++ interface (e.g., `multiply, solve`, respectively). New C and Fortran APIs will also give better direct access to SLATE's features and Matrix objects, without requiring use of our ScaLAPACK compatibility interface.

6. **Non-symmetric eigenvalue problem:** Computing the non-symmetric eigenvalue problem is significantly more computationally expensive than the Hermitian eigenvalue problem. Our implementation will leverage the latest advances, such as aggressive early deflation, to achieve high performance.

[6] http://www.icl.utk.edu/publications/series/swans.

4.3.18 WBS 2.3.3.14 ALExa

Overview The ALExa project (*Accelerated Libraries for Exascale*) focuses on preparing the ArborX, 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 the perform 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 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 multi-physics simulations in projects such as ExaStar (Tasmanian); and (4) the ability to automatically interface Fortran-based codes to existing large and complex C/C++ solftware 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) and mesh-mesh interactions such as contact in computational mechanics and solution transfer in multiphysics simulations.

Performance portable search capabilities: 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.

URL: https://github.com/arborx/ArborX

DTK (Data Transfer Kit)

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.

Mesh and mesh-free interpolation capabilities: 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.

URL: https://github.com/ORNL-CEES/DataTransferKit

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.

Sparse grids capabilities: 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: Bayesian inference; parameter estimation/calibration; model validation. global optimization and optimization under uncertainty.

URL: http://tasmanian.ornl.gov

ForTrilinos (Fortran for Trilinos)

Purpose: ForTrilinos provides a seamless pathway for large and complex Fortran-based codes to access Trilinos. In addition, the developed SWIG/Fortran allows automatic generation of the interfaces to any C/C++ library for seamless interaction with Fortran application.

Significance: The Exascale Computing Project (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. Trilinos is primarily written in C++, including its user interfaces. While C++ is advantageous for gaining access to the latest programming environments, it limits Trilinos usage via Fortran.

SWIG capabilities: The SWIG/Fortran interface generator, based on the original SWIG, creates the object-oriented Fotran wrapper code that users can access directly. This language translation will occur in both directions, allowing, for example, an inversion of control functionality in ForTrilinos that enables custom extensions of the Trilinos solvers that are Fortran-based. Once the ForTrilinos project is complete, a functional, extensible suite of capabilities to access Trilinos on next-generation computing systems will be provided.

URL: https://github.com/trilinos/ForTrilinos

Key Challenges

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.

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: Complex models usually have significant variability in execution time for different model inputs, which leads to massive down-time when employing the standard fork-join adaptive sparse grid algorithms. After the surrogate has been constructed, collecting the samples for statistical analysis (or multi-physics simulations) requires a massive number of basis evaluations and many sparse and dense linear operations.

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.

Solution Strategy

ArborX: ArborX builds on a Kokkos+MPI 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..

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: Implement asynchronous DAG-based sparse grids construction methods that preserve the convergence properties of the fork-join algorithms but are insensitive to fluctuations in model simulation time. Port the basis evaluations and linear algebra to the GPU accelerators, and leverage the SLATE/MAGMA capabilities to ensure performance portability across relevant platforms.

ForTrilinos: Develop a Fortran module for the Simplified Wrapper Interface Generator (SWIG) utility [178]. SWIG is used to parse C++ code and generate wrapper code, and was already used for this purpose for several dozen target languages, most notably Python. The project took the approach of adding a Fortran 2003 wrapper generator for SWIG in order to fulfill many of the critical feature requirements for ForTrilinos. The developed SWIG/Fortran functionality allowed to proceed with automatic generation of Fortran interfaces to selected Trilinos libraries. The work is conducted in phases, with each phase increasing the number of wrapped of Trilinos packages.

Recent Progress

ArborX: Extensive optimization work has yielded significant performance improvements on accelerated and heterogeneous architectures. Recent results on Summit show effective use of the Power9 hyperthreading capability as well as excellent performance on the V100 GPU.

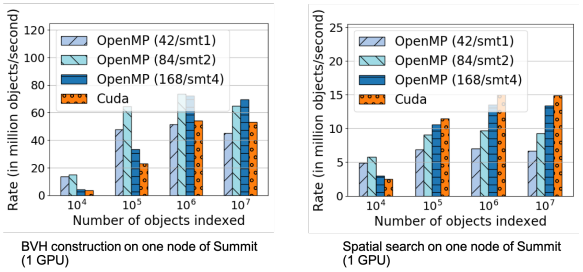


Figure 56: ArborX search performance on a single Summit node. One V100 GPU gives similar performance as the entire Power9 CPU performance on a node.

DTK: Work with partner application ExaAM (WBS 2.2.1.05) created a preliminary multiphysics driver capability for additive manufacturing simulations using a parallel-in-time coupling strategy.

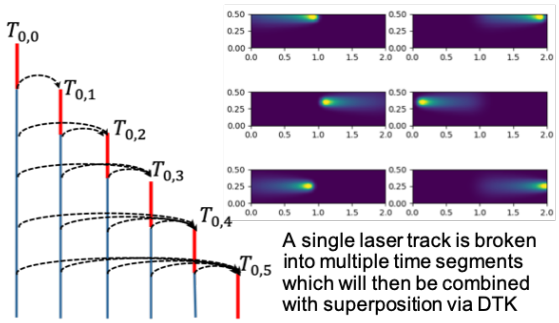


Figure 57: New ExaAM parallel-in-time coupling. Each arrow represents a DTK transfer between different grids at different time points. Thousands of DTK maps will be used to interpolate and communicate information between steps. Image courtesy of Matt Bement (ExaAM Team)

Tasmanian: The infrastructure of Tasmanian has been upgraded to support the broader ECP focus of the work. GPU acceleration of sparse grid surrogates has been implemented. Tasmanian recently enabled the ExaStar project to reduce the size of a large-memory table of neutrino opacities by 10X while still preserving accuracy.

ForTrilinos: ForTrilinos was updated to support nonlinear solvers. The list of backends supported by ForTrilinos was expanded to include serial, OpenMP and CUDA. A thorough unit testing suite was added.

Next Steps

ArborX: Continue performance engineering campaign and extend to distributed search capabilities and communication and deploy in a variety of applications.

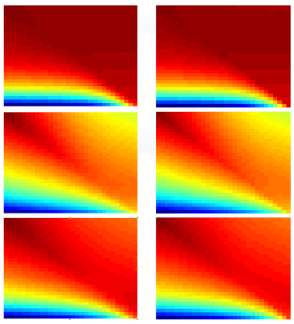


Figure 58: Tasmanian approximation (right) of neutrino capacities (left).

DTK: Support parallel-in-time coupling in the ExaAM project for simulations of metal powder bed additive manufacturing.

Tasmanian: Work will continue with the development of mixed precision algorithms for fast surrogate evaluations and integrativ the capability within the ExaStar project.

ForTrilinos: the next efforts will include

1. **Integrate developed capabilities into applications:** E3SM-MMF is an Earth system model development and simulation project. It relies on Trilinos for its implicit capabilities. The ForTrilinos project will integrate the developed nonlinear solver with IoC into E3SM-MMF to provide path forward to heterogeneous stack.

2. **Integrate ForTrilinos with upstream Trilinos project:** ForTrilinos will develop a support model to interact with upstream Trilinos developers. This will allow a faster integration of new capabilities, and more robust testing infrastructure.

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
- Workflow and provenance infrastructure – facilitating execution of complex computational science processes and the capture and management of information necessary to interpret and reproduce results
- 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:

- 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; and
- 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

Particularly in the area of DM, 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:

- Community/Coordination – designed to build the R&D community, inform ourselves and the community regarding activities in the area, track progress, and facilitate coordination.
- Targeted R&D – filling gaps in critical technology areas (storage infrastructure, workflow, provenance, data reduction and transformation, and organization and discovery).

In the workflows area, the first 3 years of the project will identify existing software systems that are in use by the DOE community and are aimed at applications that require HPC systems (eventually Exascale systems) and support further R&D to the emerging requirements of Exascale workflows as well as interaction with other parts of the software stack and adaptation to Exascale hardware architectures.

Portions of the DAV software stack are being productized and supported by industry, which 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 Mitigations 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 Hardware Integration 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.

Storage hardware 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.

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.

4.4.7 WBS 2.3.4.01 Data & Visualization Software Development Kits

Overview The Data & Visualization 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 SW Ecosystem & Delivery technical area. A description of the SDK effort can be found in Section 4.5.7.

4.4.8 WBS 2.3.4.09 ADIOS

Overview The Adaptable I/O Systems, ADIOS [179], 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 than 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 [180], and the new, object-oriented, code framework [181] 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 59, this past experience with methods and deep science engagements has led to demonstrations at leadership computing scale (on Titan and Summit).

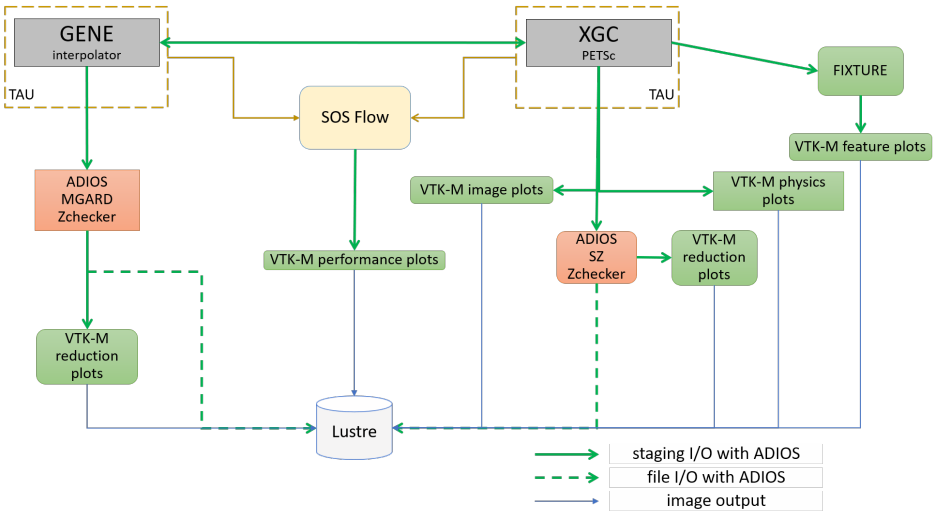


Figure 59: An example of using ADIOS to support ECP science. This sketch represents the demonstration at the February 2018 ECP Meeting, which featured WDM Fusion, CODAR, ADIOS, and other joint ECP activities. Note that all of the green arrows in the figure represent data communication or storage handled by the ADIOS infrastructure.

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 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).

Next Steps In the fourth year, the team has multiple goals: a) to tune ADIOS for Summit, evaluating its performance in ECP applications on this system, determine problems and improve the code base b) develop a performance testing framework that allows for experimentation with real applications and for easier collection of performance metrics, c) to evaluate how to incorporate ADIOS technology in the HDF5 software and d) to add ADIOS to more ECP applications, specifically targeting this year the OpenPMD particle data format and library used by WarpX.

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 *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 *STDM12-11:* This milestone presents the deliverables of three new software features in PnetCDF library: 1) making use of burst buffers to improve the I/O performance; 2) providing read capability to HDF5 files; and 3) providing read capability to BP files. The first feature aggregates I/O requests transparently to burst buffers which are later flushed to the parallel file system. This effectively achieve a better I/O bandwidth utilization. The latter two features alleviates application users from developing a separate set of I/O functions in order to read files in HDF5 and BP formats. HDF5 has been a popular file format in the scientific communities, with many third-party software packages for data analysis and visualization. In addition to improve the productivity for application scientists, these new features also encourage the PnetCDF user communities, mainly in climate research, to expand to other scientific disciplines.

STDM12-12: This milestone presents a Mercury suite release including platform-independent fault detection mechanisms for providing fault tolerance to ECP applications and data services. Additionally, a demonstration of this functionality on a production platform relevant to ECP and integration of correctness testing for this functionality into Mercury suite regression testing.

STDM12-13: This milestone presents a Darshan software release including fine-grained I/O tracing capabilities, with a mechanism to automatically enable tracing at runtime. Additionally, we demonstrate the utility of this new automatic tracing mechanism for gaining insights into an ECP application. The new trace triggering mechanism provides Darshan users more control over what files Darshan traces at runtime. Specifically, Darshan now allows users to expose a file describing a set of trace triggers that control whether or not tracing should be enabled for a particular file. Currently supported triggers include static triggers that use filename-based or MPI rank-based regex matching (i.e., filenames with a specific extension or directory prefix, or files opened by a particular range of ranks), as well as dynamic triggers based on access characteristics to files at runtime (i.e., trace all files that had at least 50 percent accesses that were small or unaligned).

STDM12-14: In this milestone we demonstrate new two-phase I/O strategies in ROMIO. We took a two-pronged approach to meeting this milestone. ROMIO has been a fundamental component of the HPC I/O stack for two decades. The original "two-phase collective buffering" optimization has served us well, but is beginning to show some design challenges on newest architectures and scales. We also developed a service-based I/O provider, through which we can deliver and explore I/O strategies in MPI and other contexts. Some ROMIO changes could be incorporated directly into MPICH and will be available in the upcoming MPICH-3.3 release. Other changes are awaiting review before inclusion. Benvolio our service-oriented I/O provider, is running on Theta, Argonne's pre-exascale machine. We have developed a ROMIO driver , allowing us to evaluate Benvolio with standard MPI-I/O benchmarks such as IOR.

STDM12-21: Delivery of composable data services for the ParSplice ECP EXALT application including

source code changes to ParSplice and demonstration of use including performance analysis on an HPC system. The Parallel Trajectory Splicing (ParSplice) application uses a novel time-parallelization strategy for accelerated molecular dynamics. The ParSplice technique (and associated application) enables long-time scale molecular dynamics (MD) simulations of complex molecular systems by employing a Markovian chaining approach allowing many independent MD simulations to run concurrently to identify short trajectories called "segments" that are then spliced together to create a trajectory that spans long time scales. A master/worker approach is used to generate segments starting from a set of initial coordinates stored in a key/value database. From these initial coordinates the workers use traditional MD simulation to generate a new segment and upon completion stores the final coordinate of the segment in a distributed key/value database. Segments are keyed by their atomic coordinates (corresponding to local energy minima) while the value is simulation state information associated with this atomic coordinate. Our work included the integration of a number of composable data services developed within the Mochi ASCR project .

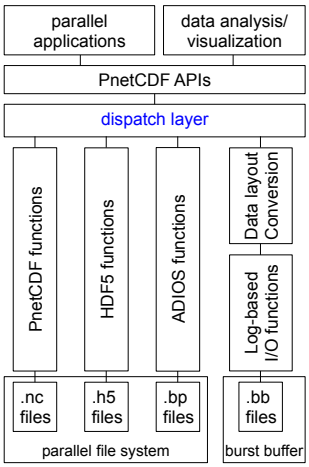


Figure 60: The new PnetCDF dispatch layer provides flexibility to target different back-end formats and devices under the PnetCDF API used by many existing applications.

Next Steps Our plan for FY20 includes:

- Augmentation of our Darshan tool to capture API use at an HDF5 dataset level of detail, building on existing (limited) HDF5 instrumentation. This enhancement will enable a better understanding of codes using the HDF5 API across all domains, applications, and platforms, facilitating more targeted and prioritized responses to performance and/or reliability challenges.
- Initial performance and overhead characterization using specific ECP application workloads. Combined with detailed data from Darshan, this activity will prepare us to best address ECP needs and also help validate our initial approach. Users will be prioritized by potential impact (e.g., supporting AMReX would impact multiple code teams), prioritization by facilities (e.g., HACC is also an early science code at ALCF), and ECP management input.
- Design of an HDF5 VOL plug-in targeting ECP code requirements and a specific platform and file system. The back-end file layout used in HDF5 is a known performance problem. We are confident that we can begin to develop an alternative back end using modern parallel I/O concepts while simultaneously performing the more comprehensive assessment described above.
- Further development and extension of FlameStore DNN model store for CANDLE application.
- Development of Quantaii/Mochi to efficiently store, organize, and index FleCSI data structures for ATDM/Ristra.

- Exploration of E3SM HDF5 code paths and optimizations for key checkpoint and analysis workloads.
- Establishment of videoconferences with consumers of Mochi technologies. This will enable us to more quickly respond to needs of our stakeholders within ECP and vendors.
- Increasing stakeholder input on prioritization of Mochi technologies and functionality. We will leverage the above mentioned videoconferences as a tool for gathering stakeholder input on our Mochi development activities, and we will adjust our plan going forward in response.
- Integration of Darshan logging into HDF5 implementation as needed, with any necessary modifications submitted to THG.
- Extension of regular testing of Mochi tools to SummitDev test platform.

4.4.10 WBS 2.3.4.13 ECP/VTK-m

Overview The ECP/VTK-m project is providing the core capabilities to perform scientific visualization on Exascale architectures. The ECP/VTK-m project fills the critical feature gap of performing visualization and analysis on processors like graphics-based processors. The results of this project will be delivered in tools like ParaView, VisIt, and Ascent as well as in stand-alone form. Moreover, these projects are depending on this ECP effort to be able to make effective use of ECP architectures.

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 such 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 like hyper threading and vector operations require even more parallel processing to leverage each core's full potential.

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.

Key Challenges 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. 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 [182, 183]. However, this decomposition works best when each processing element has on the order of a hundred thousand to a million data cells [184] and is known to break down as we approach the level of concurrency needed on modern accelerators [185, 186].

DOE has made significant investments in HPC visualization capabilities. For us to feasibly update this software for the upcoming Exascale machines, we need to be selective on what needs to be updated, and we need to maximize the code we can continue to use. Regardless, there is a significant amount of software to be engineered and implemented, so we need to extend our development resources by simplifying algorithm implementation and providing performance portability across current and future devices.

Solution Strategy The ECP/VTK-m project leverages VTK-m [187] to overcome these key challenges. VTK-m has a software framework that provides the following critical features.

1. **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 [188].

2. **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 [189]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [190, 191, 192].

3. **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 [193] and array wrappers that can adapt to target memory layouts [194].

Even with these features provided by VTK-m, we have a lot of work ahead of us to be ready for Exascale. Our approach is to incrementally add features to VTK-m and expose them in tools like ParaView and VisIt.

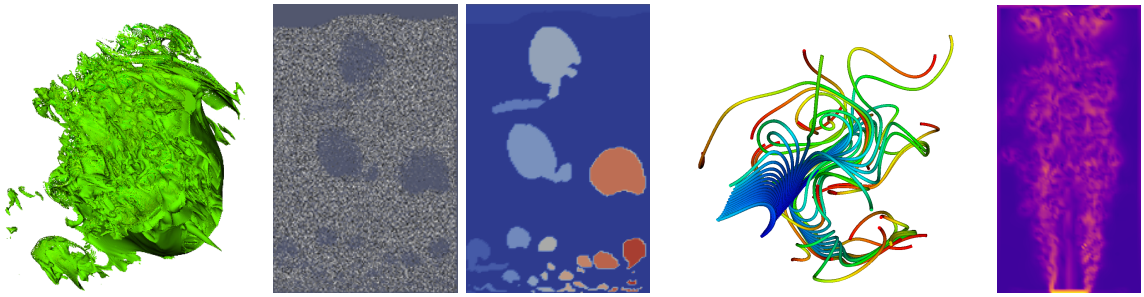


Figure 61: Examples of recent progress in VTK-m include (from left to right) clipping to form an interval volume, connected components to identify bubbles of low density, advection of particles through time, and a generated FTLE field to identify Lagrangian coherent surfaces.

Recent Progress The VTK-m project is organized into many implementation activities. The following features have been completed in the FY19 fiscal year.

- **VTK-m Releases:** VTK-m 1.3 was released in November 2018. VTK-m 1.4 was released in June 2019. VTK-m 1.5 was released in October 2019.
- **ZFP** Working in collaboration with the ZFP project (now part of WBS 2.3.4.16), the ZFP compression algorithm [195] is implemented in VTK-m. This implementation ports across ECP platforms.
- **Clipping:** A common visualization operation for extracting regions of interest, clipping provides mesh cutaways and interval volumes as demonstrated in Figure 61.
- **Merge Points:** It is often necessary to collect together positions in 3-space that are nearby [196].
- **Connected Components:** It can be physically significant to identify groups of elements that are mutually connected by either topology or common field attributes as demonstrated in Figure 61.
- **Particle Advection:** Many flow visualization algorithms depend on computing the movement of weightless particles in a flow vector field, which may change over time [197]. These include stream lines, path lines, Lagrangian coherent structures, and stream surfaces as demonstrated in Figure 61. We have also added tube geometry to improve rendering representation.
- **Mesh Quality:** Cell metrics provide important physical and meta information about a geometry.
- **Surface Normals:** Surface normals are an important feature for proper representation in rendering. Ensuring consistent ordering is important to avoid rendering artifacts.
- **Lightweight Cell Library:** To help consolidate common code between VTK-m and other software, the code for cell management has been extracted into its own lightweight library.

Next Steps Our next efforts include:

- **Performance Regression Testing:** It is important to ensure that in addition to being correct, the performance of VTK-m code does not regress.
- **Kokkos Devices:** We will explore the feasibility of using the Kokkos libraries to implement the device porting layer.
- **Higher Order Meshes:** Many ECP simulations use high order interpolation techniques on their meshes.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND 2019-13411 R

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.

VeloC has been released and shows significant lower checkpointing overhead for several ECP applications, such as HACC, LatticeQCD, EXAALT. VeloC is a next generation checkpointing system that builds on SCR, which won the prestigious R&D 100 award in 2019.

Key Challenges VeloC faces several key challenges:

- **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 vendors 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 immediately after a failure is more efficient than submitting a new job (which may wait in the batch queue) that restarts from the latest checkpoint.
- **Portability and robustness:** applications need to run on a variety of supercomputing architectures, each featuring distinct capabilities. Their critical data structures that need to be checkpointed are constantly growing in size and complexity. Therefore, a flexible checkpointing solution is needed that can adapt to a variety of scenarios and configurations without sacrificing performance and scalability.

Solution Strategy To address these challenges, VeloC adopts the following principles:

- **Multi-level checkpointing:** 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.
- **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.
- **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 plugin-able modules. This simplifies the implementation of the asynchronous mode, enables users the flexibility choose any combination of resilience strategies, as well as to customize the checkpoiting 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. The integration with HACC aimed to isolate the checkpointing code in a plugin to enable easier maintenance, ability to switch checkpointing on/off and sharing of critical data used for checkpoints with other in-situ plugins (e.g., analytics, post-processing, etc.). To this end, we designed and implemented a VELOC checkpointing plugin for CosmoTools, the in-situ framework used by HACC. For LatticeQCD and EXAALT, the integration has been performed directly into the main code. Integration with other ECP applications is ongoing.

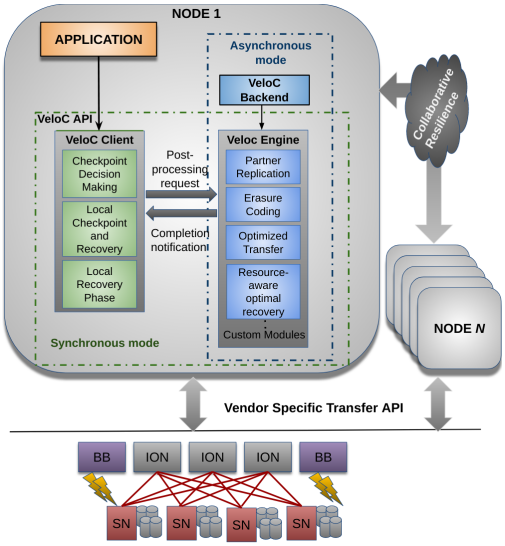


Figure 62: VeloC: Architecture

In parallel with the co-design effort done in collaboration with the ECP application teams, we improved the asynchronous mode mentioned previously and illustrated in Figure 62. Specifically, we have introduced an experimental hybrid local checkpointing strategy that leverages hybrid local storage (memory, flash) simultaneously to enable faster asynchronous flushes to the parallel file system [198]. Furthermore, we have explored interference mitigation strategies during asychronous checkpointing based on the prediction of shared resource utilization, which can be used for better scheduling of background operations [199].

The latest release v.1.2 introduces several new capabilities: (1) non-collective mode that enables processes to checkpoint independently rather than as a single group; (2) improved file-based mode that preserves custom file names when flushing to the parallel file system, which is especially useful for applications that make use of the SCR-to-VeloC compatibility translation library; (3) AXL, the library responsible for transfers to/from external storage now auto-detects and configures the best way to transfer files between local and external storage.

In addition, we have added several new features that facilitate better integration with the ECP ecosystem: (1) restart-in-place scripts for the following platforms: ANL Theta, ORNL Summit, LLNL Lassen; (2) automated testing infrastructure based on Travis, which facilitates a smooth transition towards the ECP continuous integration initiative; (3) Python bindings, which enables the use of VeloC in applications that make use of high level analytics and artificial intelligence libraries (e.g. Keras); (4) Spack installation packages and integration into the OpenHPC distribution.

Finally we expanded the documentation and created tutorials that were presented at various international venues to raise awareness about VeloC within the broader HPC community at international level.

Next Steps We are working towards several goals: (1) improving the integration of VeloC with resource managers to automate the process of checking the status of jobs and nodes, detect failures and relaunch jobs on failed nodes; (2) improving portability and robustness by introducing support for flexible client-engine communication (e.g. Mercury) and advanced serialization (e.g. C++ high-level data structures); (3) continue hardening the integration with existing ECP applications; (4) improve the automated testing infrastructure and build environment for VeloC to support comprehensive testing on ECP machines.

In parallel, we will continue to collaborate with the application teams to address new requirements should they arise. Furthermore, we will expand our user base beyond ECP to interact and get feedback from the broader international community.

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. Current lossless compression methods suffer from low compression ratio and do not adequately address the limitations in storage bandwidth and storage space of projected exascale systems. Existing lossy compressors are not covering the needs of many ECP applications.

The VeloC-SZ project is extending and improving the SZ lossy compressor for structured and unstructured scientific datasets respecting user-set error controls. 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 improve to support all data types used in ECP applications and I/O libraries, (iii) integrating and optimizing the integration of SZ in ECP client applications, (iv) porting and optimizing SZ for Aurora and Frontier, (v) developing automatic compression parameter tuning, (vi) delivering a comprehensive test suite and extensively testing SZ and its different implementations for all client ECP applications. Our goal is to produce a high-quality lossy compressor responding to the needs of ECP exascale applications and experiment users. To this end, 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:

- 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.
- Another challenge is supporting the compression optimization for a large variety of data formats. A template based approach must be used to improve robustness, debugging and testability.
- A third 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. NWChem-X integrates SZ for checkpoint/restart.
- Optimization of SZ for Aurora and Frontier requires writing portable accelerator codes that are non trivial for complex compression pipeline.
- While automatic compression parameter tuning to maximize compression performance (speed and accuracy) is useful for end-users to avoid a cumbersome optimization process, is not trivial to implement.
- 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.

Solution Strategy As for the first challenge, we keep a close communication with ECP application users to understand their specific demands on the lossy compression. For instance, we have a weekly meeting with ECP application teams to discuss the required error bounds and compression speed and quality. We also provide multiple types of error bounds (such as absolute error bound, PSNR, relative error bound) allowing users to control the errors in different ways. We also exploit an adaptive prediction method to optimize the compression quality for diverse datasets.

As for the second challenge, we refactor SZ in C++, starting from the current C version. This refactoring is the perfect occasion to implement a new more modular design of SZ, capable of integrating more stages in the compression pipeline and of selecting compression stages based on specific application data features.

As for the third challenge, our weekly meetings with the application teams provide us a clear understanding of the integration pathway as well as the expected performance. We also often discuss potential solutions with application teams when needed.

Concerning the fourth challenge, we are in contact with ALCF and OLCF as well as with vendors to access simulators and early systems that will help to optimize the aceelerator implementations.

For the fifth challenge, we are designing and integrating of a control loop capable of adjusting compression parameters from user set constraints. The automatic part of the tuning will use optimization techniques to exploration of a large potential configuration space.

Concerning the test suite, we are continuously developing and improving it. We will need to adapt it for C++ as part of the SZ refactoring. We will also use ECP testing environment when it becomes available.

Recent Progress SZ is an open source software on github, and the latest version is SZ 2.1.7, which has been optimized compared with SZ 2.0.

SZ 2 also has significant improvement on compression ratio compared to SZ 1, especially on high-compression cases. Specifically, SZ 2 adopts a hybrid prediction model by leveraging both Lorenzo predictor and linear-regression prediction method. It splits each dataset into multiple non-overlapped blocks and selects the bestfit prediction approach in each block by a sampling method. Figure 63 demonstrates the core step (data prediction and linear-scaling quantization) in SZ lossy compression using a 2D dataset and that SZ still produces a high visual quality for the NYX VX field with a compression ratio of 156:1.

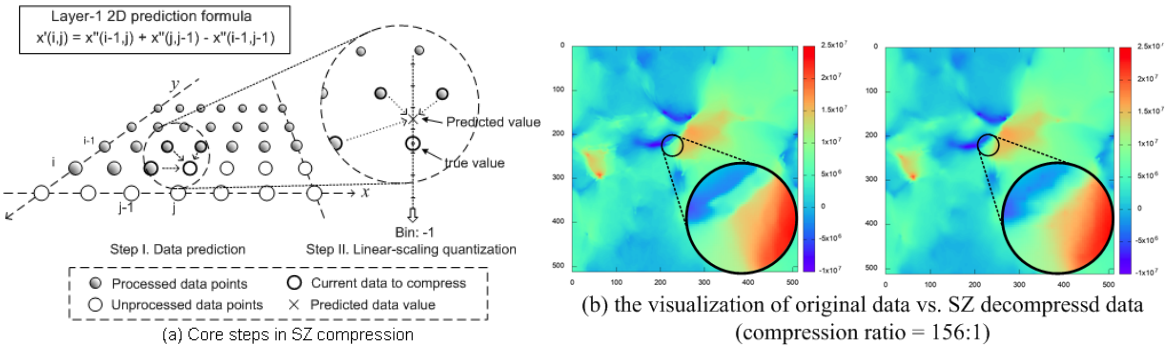


Figure 63: SZ principle and original vs. decompressed NYX VX field

We have improved the compression quality for different ECP applications significantly, including ECP HACC, EXAFEL, GAMESS, and others. For instance, SZ leads to higher compression ratio for asbolute error bound on these datasets than the second best lossy compressor, with comparative compression rate. We also implemented effective compression method supporting point-wise relative error bounds for the ECP ExaSky project. Experiments with point-wise relative error bound based compression shows that our solution leads to 31%-210% higher compression ratio than other lossy compressors do (best paper at IEEE CLUSTER18). We accelerated the compression rate significantly (by 50% in most of cases) by a table-lookup method, which was published in IEEE MSST19. Moreover, SZ allows users to customize their own prediction method to adapt to the special features of datasets. For instance, the compression ratio has been improved about 2∼3X by leveraging the patterns existing in the GAMESS dataset (overall best paper award at IEEE Cluster 2018).

All the improvements and functionalities developed in SZ comes from user practical requirements. SZ supports multiple I/O libraries such as HDF5, PnetCDF, and ADIOS; and it also supports both C and Fortran. We also developed various parallel versions such as multi-threaded version and GPU version. SZ also supports random access during the data decompression, allowing users to decompress only interesting regions/parts of the data. We also optimized the I/O performance by exploring the best tradeoff between the compression ratio and compression rate, also taking into account the varied bandwidth with increasing execution scales. This work was published in IEEE Cluster 2019.

Next Steps Our next efforts are: Improve compression performance by analyzing SZ's performance, refactoring SZ in C++, integrating SZ in more ECP applications, and implementing a portable GPU version for Aurora and Frontier and improving SZ testing environment.

4.4.13 WBS 2.3.4.15 *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 this project, we have developed various HDF5 features are in development to address efficiency and other challenges posed by data management and parallel I/O on exascale architectures. The ExaIO-HDF5 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 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 related data movement in exascale systems, full single-writer / multi-reader (SWMR) for workflows, asynchronous I/O, querying data and metadata, and scalable sub-file I/O.

Many of the funded exascale applications and co-design centers require HDF5 for their I/O, and enhancing the HDF5 software to handle the unique challenges of exascale architectures will play an instrumental role in the success of the ECP. For instance, AMReX, the AMR co-design center, is using HDF5 for I/O, and all the ECP applications that are collaborating with AMReX will benefit from improvements to HDF5. The full SWMR feature will support the needs of ExaFEL's workflow in appending data incrementally. The virtual Object Layer (VOL) and interoperability features with netCDF and ADIOS data open up the rich HDF5 data management interface to science data stored in other file formats. The project will be 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 using a hierarchy of storage devices and topology of compute nodes efficiently, 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. Adding asynchronous I/O to an application's existing ability to perform nonblocking communication is a necessary aspect of maximizing the utilization of valuable exascale computing resources.

Solution Strategy *Utilizing complex compute and storage hardware.* Data Elevator is being developed in this project to exploit multi-level storage hierarchies. The Data Elevator library intercepts HDF5 file access calls and redirects them to intermediate faster caching storage layers, which future application reads or writes will then access. When updates or writes to the intermediate data are finished, Data Elevator's server daemon moves the data transparently to its final destination on colder storage layers, such as a disk-based parallel file

system. This occurs transparently to the application, without modifying its source code or placing a burden on users to move the data explicitly to and from intermediate caching storage layers.

In our prior work, improved communication times were achieved for a broad spectrum of data movement patterns such as those seen in multi-physics codes, parallel I/O aggregation, and in situ analysis, and have also improved the time to access parallel file systems. In addition to our work on Data Elevator, the team is also developing these topology-aware optimization strategies as a Virtual File Driver (VFD), which can be plugged into HDF5.

Asynchronous I/O Virtual Object Layer (VOL) Connector: 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, we chose to perform I/O operations in 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 64, 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.

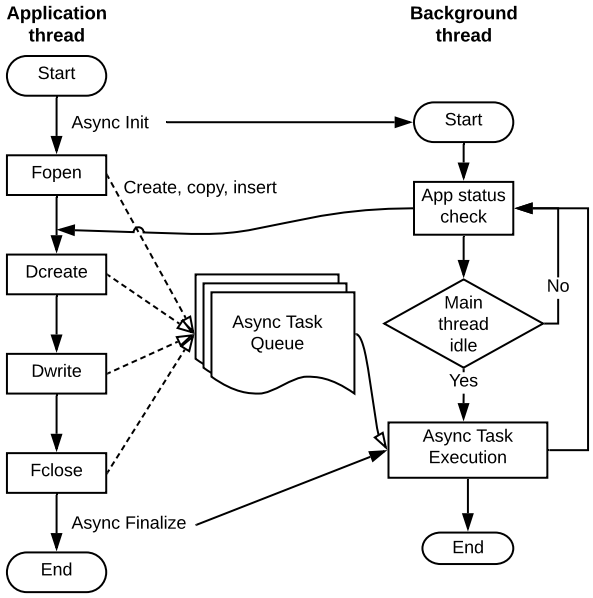


Figure 64: An overview of asynchronous I/O as a HDF5 VOL connector

Recent Progress *Prototype implementation of Data Elevator read prefetching.* The project team has developed a prototype implementation of read caching and prefetching functionality, and has tested with various data read kernels from real applications. Using the burst buffers on Cori, the Data Elevator achieves 1.2–3X performance improvement over a highly tuned HDF5 code in reading data. Performance evaluation included representative I/O of convolution on climate modeling data, gradient computation of a plasma physics data, and vorticity computation of a combustion dataset.

Integration of the VOL framework into the HDF5 develop branch. The VOL feature branch has been integrated into the main HDF5 development branch. Earlier in the project, an older VOL branch was brought in sync with the latest development branch, but this has been enhanced to allow stacking multiple VOL connectors. The development branch with the VOL feature has been tested with various VOL connector codes. A pass-through VOL connector also has been developed to test the stack-ability of multiple VOL connectors.

Supporting ECP application I/O The ExaIO-HDF5 team has been working with various applications in the ECP portfolio. Applications in the AMReX co-design 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.

Asynchronous I/O We have evaluated the proposed asynchronous I/O framework on the Cori supercomputer at the National Energy Research Scientific Computing Center (NERSC) with several benchmarks

and I/O kernels. Experimental results show that our method can effectively mask the I/O cost when the application is idle or performing non-I/O operations.

Next Steps The ExaIO-HDF5 team is designing subfiling - a strategy for reducing locking and contention on parallel file systems, developing topology-aware I/O, fine-tuning asynchronous I/O and the Data Elevator caching and prefetching functionality, and supporting ECP AD and ST teams and facilities in improving the overall performance of HDF5.

4.4.14 WBS 2.3.4.15 UnifyCR – 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 are developing UnifyFS, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. UnifyFS will address a major usability factor of current and future systems, because it will enable applications to gain the performance advantages from distributed burst buffers while providing ease of use similar to that of a parallel file system.

Key Challenges The hierarchical storage for future HPC systems will include 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 are developing UnifyFS, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. In Figure 65, 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 dumps by appliations. 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 third year of this project, the team worked to implement the design plan that was established in the first year of the project (See Figure 66). In particular we completed the replacement of MPI communication with DataLib software; we developed support for moving data into and out of UnifyFS by I/O libraries and VeloC; we implemented and tested support for VeloC; we updated and improved our

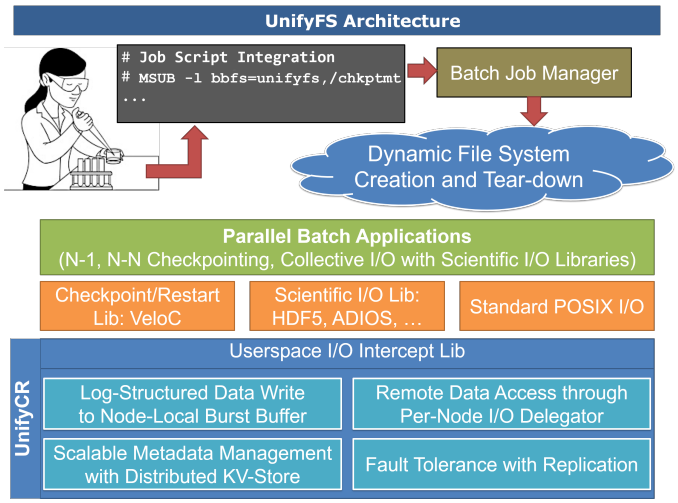


Figure 65: UnifyFS Overview. Users will be able to give commands in their batch scripts to launch UnifyFS within their allocation. UnifyFS will work with POSIX I/O, common I/O libraries, and VeloC. Once file operations are transparently intercepted by UnifyFS, they will be handled with specialized optimizations to ensure high performance.

integration with resource managers; we furthered our efforts to evaluate and implement support for I/O libraries. Additionally, we detected and fixed numerous bugs, improved documentation, and implemented a metadata abstraction layer, and released UnifyFS 0.9.

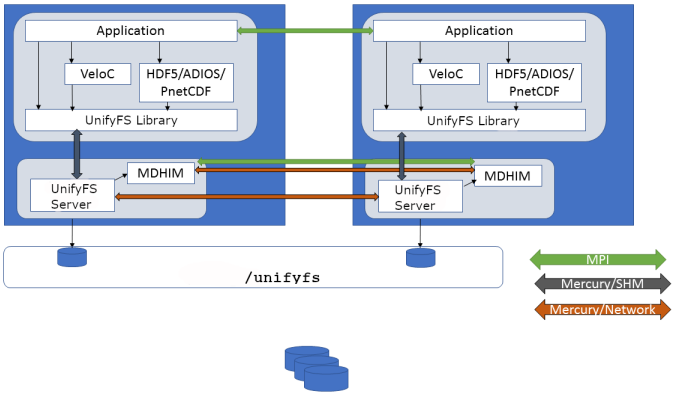


Figure 66: UnifyFS Design. The UnifyFS instance consists of a dynamic library and a UnifyFS daemon that runs on each compute node in the job. The library intercepts I/O calls to the UnifyFS mount point from applications, I/O libraries, or VeloC and communicates them to the UnifyFS daemon that handles the I/O operation.

Our source code for UnifyFS is available on GitHub at https://github.com/LLNL/UnifyFS.

Next Steps For our next milestone effort, we are focused on delivering a design for the UnifyFS API for use by HDF5 and other I/O libraries. Additionally, we are continuing work on integrating and supporting ECP collaborator software including: I/O libraries HDF5, MPI-IO, PnetCDF, and ADIOS; improving support for ECP VeloC; and targeting integration with ECP applications such as E3SM, GEOS, and Chombo. We also plan to release UnifyFS 1.0 early in the 2020 calendar year.

4.4.15 WBS 2.3.4.16 ALPINE

Overview ECP ALPINE/ZFP will deliver in situ visualization and analysis infrastructure along with lossy compression for floating point arrays to ECP Applications.

ALPINE developers come from the ParaView [200, 201] and VisIt [202] teams and ALPINE solutions will deliver in situ functionality in those tools, as well as through Ascent [203], 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 performant in situ algorithms. The challenge is that our existing infrastructure tools need to be made 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 VisIt, including their in situ libraries Catalyst [204] and LibSim [205], and then to integrate and augment them with ALPINE 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 Office of Science (SC) and the DOE National Nuclear Security Agency (NNSA). These two products provide significant coverage of ECP Applications, and we can leverage their existing engagements to deliver ALPINE's algorithms and infrastructure. We are 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, VisIt, and Ascent. Thus all development effort by ALPINE will be available in all of our tools. Further, by leveraging VTK-m, we will be addressing issues with many-core architectures. Figure 67 illustrates our software strategy.

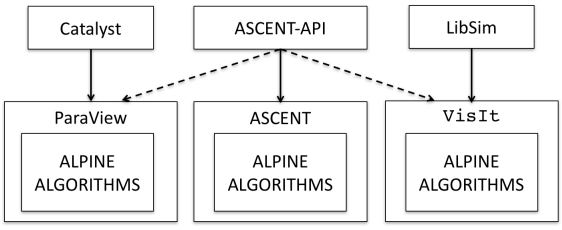


Figure 67: ALPINE's strategy for delivering and developing software. We are making use of existing software (ParaView, VisIt), but making sure all new development is shared in all of our tools. The dotted lines represent ongoing work, specifically that the Ascent API will work with ParaView and VisIt.

ALPINE is developing a suite of in situ algorithms designed to address in I/O and data output constraints. We have four mature algorithms.

Lagrangian analysis of 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 [206, 207, 208].

Topological analysis can be 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 [209].

Adaptive sampling can be used to guide visualizations and extracts to the most important parts of the simulation, significantly reducing I/O. Figure 68 shows an adaptive sampling technique based on importance to preserve features in the data [210, 211, 212].

Moments-based pattern detection can be used to find rotation-invariant patterns [213, 214, 215].

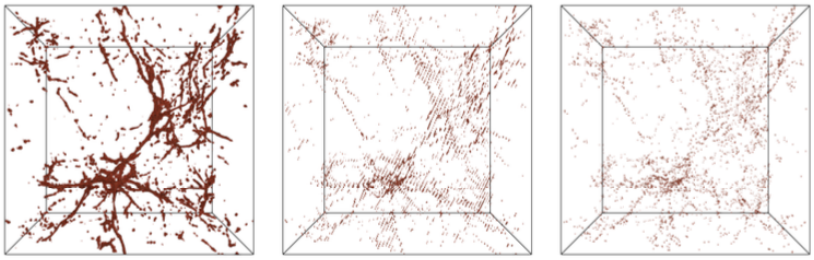


Figure 68: Point rendering results from Nyx simulation using (left to right): ALPINE adaptive sampling (sampling ratio 0.5%); regular sampling (sampling ratio 1.5%); random sampling (sampling ratio 0.5%).

Recent Progress In recent infrastructure work, we have integrated VTK-m into VisIt and ParaView (STDA04-32) and ALPINE infrastructure has been fully integrated into ParaView and VisIt to support ALPINE algorithms to run both on CPUs and GPU accelerators. ParaView visualization support was added via Ascent Python Extracts.

For algorithms, we have completed the initial R&D phase for our four mature algorithms (STDA04-5). To address a wider range of application needs, ALPINE is expanding its suite of algorithms. These new algorithms are in the development and prototyping stage:

Scalable Statistics can be used to summarize data and timesteps in situ. These basic VTK-m based filters can move post-processing steps in situ or, e.g. to trigger saving specific timesteps.

Tracking over time analyzes multivariate simulation data sets and explores temporal relationships of variables based on important scientific features. Statistical data modeling techniques represent simulation data in situ and both feature-driven and feature-agnostic techniques can be utilized for analysis purposes.

Automated Viewpoint Selection can be used to select optimal viewpoints for visualizations in situ.

Feature extraction 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 [216] and execute it using different runtimes (e.g., MPI, Charm++, Legion).

In recent milestones, we integrated the team's mature and emerging algorithms into the ALPINE infrastructure (STDA04-32, STDA04-41, STDA04-43) and demonstrated early in situ implementations in ECP codes (STDA04-33) with demonstrations of parallel distributed versions for each mature algorithm (STDA04-37). Emerging algorithms have been prototyped with ECP applications. In a final FY19 milestone for these algorithms, we demonstrated end-to-end integration pipelines with ECP Applications and ALPINE infrastructure for other in situ algorithms (STDA04-38):

- ExaSky:Nyx was integrated with Ascent, running adaptive sampling and outputting a Cinema database; demonstrated on Summit with ALPINE codes on the GPUs and Nyx on the CPUs.

- EQSIM:SW4 was integrated with Ascent, running the Lagrangian algorithm; demonstrated on Summit GPUs.
- WarpX was integrated with Ascent and accessed the Moments-based pattern detection algorithm through ParaView; demonstrated on Summit CPUs.
- PeleC was integrated with Ascent, running adaptive sampling; demonstrated on Summit CPUs.

Next Steps Plans for FY21-23 will continue the focus of integration and delivery to ECP applications. The emphasis in 2020 and 2021 will be porting the team's software products to Frontier and Aurora, doing initial performance studies relevant to ECP applications. The team will also focus on outreach and prototyping integrations with ECP application codes in order to facilitate full integration in later years.

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, 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 while ensuring thread safety; 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**. 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 and CUDA 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 Work on Fortran and Python bindings to ZFP's high-level compression API has been completed (STDM13-16). We have extended ZFP to support lossless compression (STDM13-15) and have continued to explore *preconditioning* approaches to use with particle and unstructured data (STDM13-24). ZFP relies on spatial correlation in structured data for compression and expanding its use beyond structured data requires strategies to make the data amenable to compression. We experimented with local sorting approaches but found them largely ineffective. A more successful preconditioner is to pass structured data through a wavelet transform, which we found to improve accuracy by as much as an order of magnitude for the same storage cost. We will continue to explore ways to expand ZFP's usability for particles and unstructured datasets. Finally, we have implemented data-dependent variable-length storage arrays (Figure 69) that support random access (STDM13-34) and are finishing up related efforts on read-only arrays (STDM13-33).

The results of our R&D efforts have been documented through publications [217, 195, 218, 219], 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 [220, 221, 222, 223, 224]. We have an upcoming tutorial at SC19 with SZ [225]. Lastly, ZFP version 0.5.5 has been released [226].

Next Steps The next year will focus on two primary efforts: (1) refactoring our compressed-array code to reduce redundancy, improve robustness, and facilitate further development; and (2) filling gaps in a large and sparsely populated feature space, including missing language bindings. We will continue outreach to ECP applications and ECP software technologies.

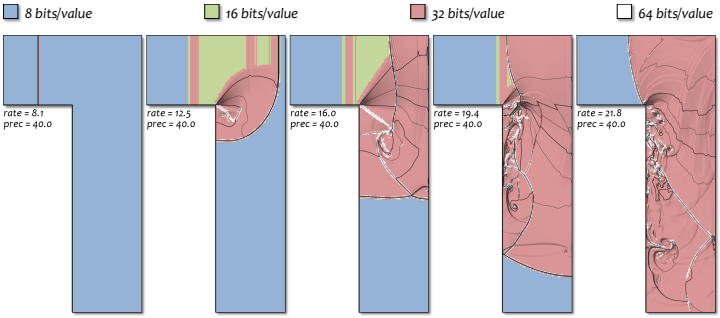


Figure 69: ZFP variable-rate arrays spatially adapt bit allocation.

4.5 WBS 2.3.5 SW 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 (Extreme-Scale Scientific Software Stack, or E4S) in multiple packaging formats technologies – from-source builds, modules, and containers
- Oversight of the ST SDKs (Software Development Kits) 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 the Facilities) for use by ECP teams at the Facilities
- Development and hardening of methods for software deployment through the use of container technology
- 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 Hardware Integration L3 area for Deployment of Software on 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 is gaining 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 to 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, 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 0.2 of the Extreme-Scale Scientific Software Stack (E4S) was released in January 2019 comprising of a subset of ST projects that had Spack packages. This release also demonstrated the use of container technologies, with inclusion of Docker, Singularity, Shifter, and CharlieCloud containers for people to use a starting point for integration into applications. In November 2019, version 0.3 of the E4S will be released and we will follow a regular cadence of E4S releases, with ever-increasing number of ST products included, 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 2020 and beyond, we plan to further define these SDKs and their community policies, and develop a delivery and deployment mechanism that will get these products into the hands of our application users.

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. 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.

In 2020, we plan to fill a gap in the ST portfolio with regard to scientific workflows and are working to determine the extent of the gap and to identify a technical plan to fill it. This plan will be evaluated by the ST leadership with regard to merits of the technical plan, potential to have an impact on applications by the end of the ECP, deployability on the exascale machines, and sustainability beyond the end of the ECP.

4.5.5 Risks and Mitigations Strategies

- Deploying E4S on unknown architectures – use Spack for deployment to decrease installation complexity
- Keeping updated versions of ST and dependent software in synch 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 hard to come to agreement on a “common” looking solution; may not be possible to generate containers that are both portable and performant
- OpenHPC partnership is ill-defined, and unfunded
- Sustainability of ECP ST capabilities after ECP has ended

4.5.6 Future Trends

Software development kits 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.

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.

We also believe that 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 or faster than running the application on bare metal. Container-based scientific workflows will also 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 Software Development Kit (SDK) project supports a set of activities aimed at

- establishing Community Policies aimed at increasing the interoperability between and sustainability of ST software packages, using the xSDK [120] community package and installation policies [121] as a model.
- coordinating the delivery of ECP ST products through the Extreme-Scale Scientific Software Stack (E4S) [227], a comprehensive and coherent set of software tools, to all interested stakeholders on behalf of ECP ST. This includes 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 will work 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. In addition to the delivery of the E4S through Spack, the SDK will also explore the option of binary delivery, possibly through OpenHPC.

Recognizing the different release readiness and broader maturity differences between different ECP ST products, the early release strategy is 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.

Recent Progress In January 2019, E4S Release 0.2 was posted to the new external E4S website [227]. It includes a subset of ECP ST software products, and demonstrates the target approach for future delivery of the full ECP ST software stack. Also available are a number of ECP ST software products that support a Spack package, but are not yet fully interoperable. As the primary purpose of the 0.2 release is demonstrating the ST software stack release approach, not all ECP ST software products were targeted for this release. Software products were targeted primarily based on existing Spack package maturity, location within the scientific software stack, and ECP SDK developer experience with the software.

E4S release 0.2 is also available through a container release that includes support for Docker, Singularity, Shifter, and Charliecloud. The release allows ECP applications that use MPI to be released in a binary form using libraries from the container. The MPI runtime layer can be substituted during execution with the native MPI (e.g., Intel MPI, Cray MPICH, MVAPICH2) that uses the high-speed inter-node network interconnect.

The initial set of ECP ST SDKs was finalized in December 2018 after consultation with ECP ST leadership. The make up of the SDKs is expected to evolve over time, but the current definitions provide a basis for beginning to form the associated SDK communities. Figure 70 illustrates the initial division of ST products into SDKs. Note that the ecosystem group listed is currently not anticipated to become a SDK. Rather, the members of that group will be considered E4S software not associated with a specific SDK and will be responsible only for the general community policies that will apply to all ECP ST products. This is due to the large variety of software included in that group and an anticipated lack of commonality that will lead to additional beneficial community policies. Smaller-scale collaboration between these teams may be encouraged in select cases.

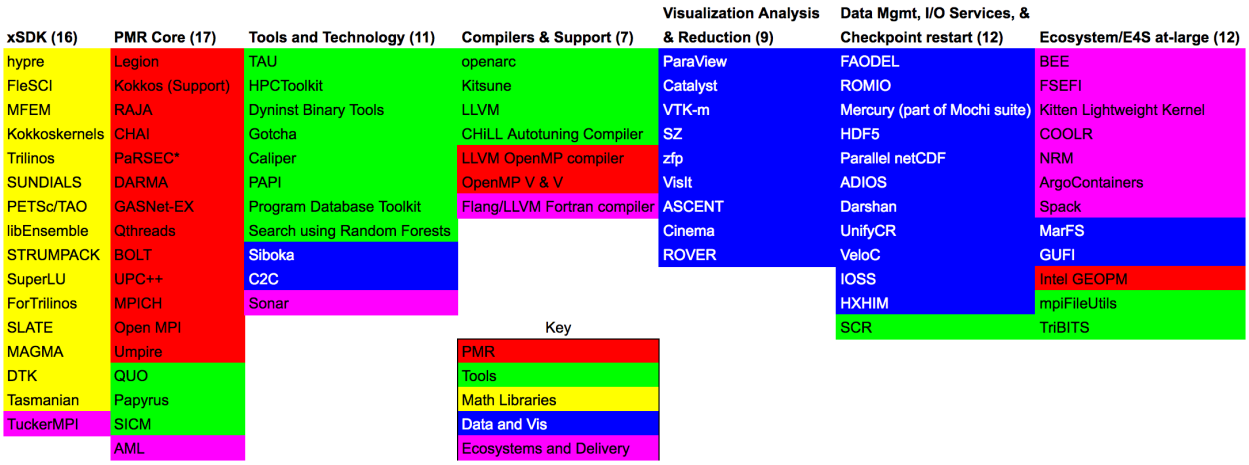


Figure 70: The above graphic shows the breakdown of ECP ST products into 6 SDKs (the first six columns). The rightmost column lists products that are not part of an SDK, but are part of Ecosystem group that will also be delivered as part of E4S. The colors denoted in the key map all of the ST products to the ST technical area they are part of. For example, the xSDK consists of products that are in the Math Libraries Technical area, plus TuckerMPI which is in the Ecosystem and Delivery technical area.

Next Steps Current and near-term efforts include:

- Defining community policies for E4S.
- Assisting with E4S deployment to computing Facilities.
- Adding additional ST software to E4S for inclusion in the version 1.0 release.
- Creating a Spack build-cache for improving the ease of installation of E4S packages.

- Creating a set of reproducible container recipes for E4S SDKs.
- Creating base images for Linux x86_64, ppc64le, and aarch64 architectures.
- Beginning SDK-specific community policy discussions within newly formed SDK communities.
- Testing new ECP-funded continuous integration capability.

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 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, filesystem, 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 (Shifter, Charliecloud, and 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 and produce a technical report to help educate new users and

developers to the advantages of containers, as well as a best-practices report 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

1. Developed a library for labeling and comparing microarchitectures for compatibility. This enables Spack to distribute optimized binaries for specific target architectures, and to label containers based on what machines they are compatible with.
2. Worked with the E4S team to build a Spack environment-based build of E4S. In contrast to prior iterations, this version is reproducible and integrates with DevOps and Spack's GitLab pipeline support.
3. Held a hackathon at LLNL for HI teams to get Spack pipelines working. Five teams were able to set up and deploy Spack pipelines.
4. Produced the first revision of our best practices document for efficient container utilization: "Containers in HPC: Best practices and pitfalls for users," hosted on GitLab pages.[7]
5. Developed an enhanced tutorial session which introduces the context of using containers in HPC systems, with a demo on NERSC's Cori system. The tutorial had over 60 participants at ISC and will appear again at SC19.

Next Steps

1. Modify Spack to support generating multi-stage container build recipes. This will enable Spack to produce the Dockerfiles required to build minimal application images.
2. Continue working with E4S on Spack build hardening, and continue to Support ST teams' Spack usage.
3. Produced the first revision of our best practices document for efficient container utilization: "Containers in HPC: Best practices and pitfalls for users," hosted on GitLab pages.[8]
4. Investigate container scalability and optimization strategies, both at the container image layers and also within container runtimes.
5. Build optimized container images which leverage new features in Spack to generate the full E4S software stack.
6. Conduct deep dives with several AD teams to aid and advance container utilization strategies for application development.
7. Continue tutorials and outreach activities.

[7] https://reidpr.gitlab.io/best-practices/
[8] https://reidpr.gitlab.io/best-practices/

4.6 WBS 2.3.6 NNSA ST

End State: Software used by the NNSA/ATDM National Security Applications and associated exascale facilities, hardened for production use and made available to the larger ECP community.

4.6.1 Scope and Requirements

The NNSA ST L3 area is new in FY20, although the projects included have all been part of the ECP in the past. The capabilities of these software products remains aligned with the other Software Technology L3 areas from which they were derived, but are managed separately for non-technical reasons out of scope for 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 are 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 (Advanced Technology Development and Mitigation) 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 along 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 Mitigations Strategies

A primary risk within this L3 area is that the 2020 ASC Level 1 milestone designed as a capstone for the ATDM initiative and a decision point for the ultimate transition of those applications into the core ASC porfolio, will fail due to the in adequacy of these software products. While not all of them are on the critical path to application success (instead focusing on productivity enhancements for end users, or analysis functionality), it is expected that first and foremost they will contribute to the success of that milestone, as any subsequent ASC milestones and decision points about the ultimate fate of those applications. Mitigation is to use other ASC funding to bolster these efforts as needed.

A secondary 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 dependant 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 dependant products are in the broader community, 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 Advanced Technology Development and Mitigation (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 https://gitlab.com/StanfordLegion/legion.

The **Kitsune** Project, part of LANL's ATDM CSE efforts, provides a compiler-focused infrastructure for improving various aspects of the exascale programming environment. At present, our efforts are primarily focused on advanced LLVM compiler and tool infrastructure to support the use of a *parallel-aware* intermediate representation. In addition, we are actively involved in the Flang Fortran compiler 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 Data and Visualization project develops scalable solutions for data analysis as part of the **Cinema** project, which provides post-facto interactive interactions with a dataset using a fraction of the file storage.

The **BEE/Charliecloud** subproject is creating software tools to increase portability and reproducibility of scientific applications on high performance and cloud computing platforms. Charliecloud [228] is an unprivileged Linux container runtime. It allows developers to use the industry-standard Docker [229] toolchain to containerize scientific applications and then execute them on unmodified DOE facility computing resources without paying any performance penalty. BEE [230] (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.

Key Challenges

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 intermediate representation 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 application with the Cinema software and the management of the volumous data from these applications.

Bee/CharlieCloud Other HPC-focused container runtimes exist, such as NERSC's Shifter [231] and Singularity [232]. 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

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.

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.

Kitsune: Given the project challenges, our approach takes aspects of today's node-level programming systems (e.g. OpenMP and Kokkos) and popular programming languages (e.g. C++ and Fortran) into consideration and aims to improve and expand upon their capabilities to address the needs of ECP. 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 breakthrough technologies.

Unlike current designs, our approach introduces the notion of explicit parallel constructs into the LLVM intermediate representation, building off of work done at MIT on Tapir [233]. We are exploring extensions to 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.

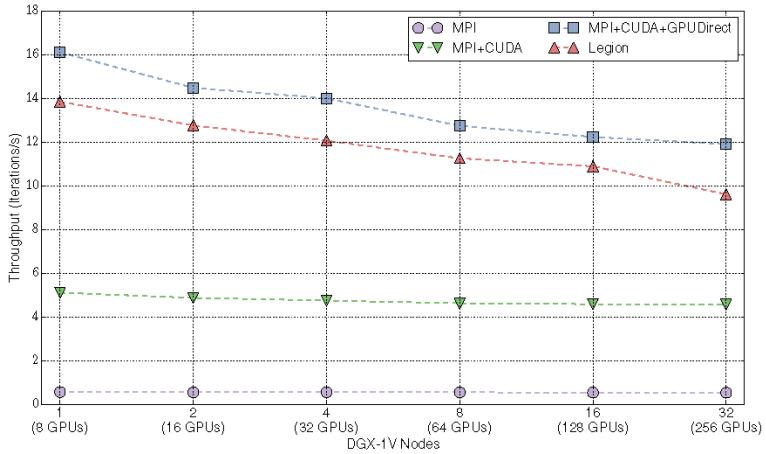


Figure 71: Productivity features such as Dynamic Control Replication scales well across multi-GPU systems in unstructured mesh computations.

Cinema: The LANL Cinema project is focused on delivering new visualization capabilities for creating, analyzing, and managing data for Exascale scientific applications and Exascale data centers.

Cinema [234] 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 viewport for later analysis and visualization by scientists. Cinema is an innovative way of capturing, storing and exploring extreme scale scientific data. Cinema is essential for ECP because it embodies approaches to maximize insight from extreme-scale simulation results while minimizing data footprint

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

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 peformance optimization activites.

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

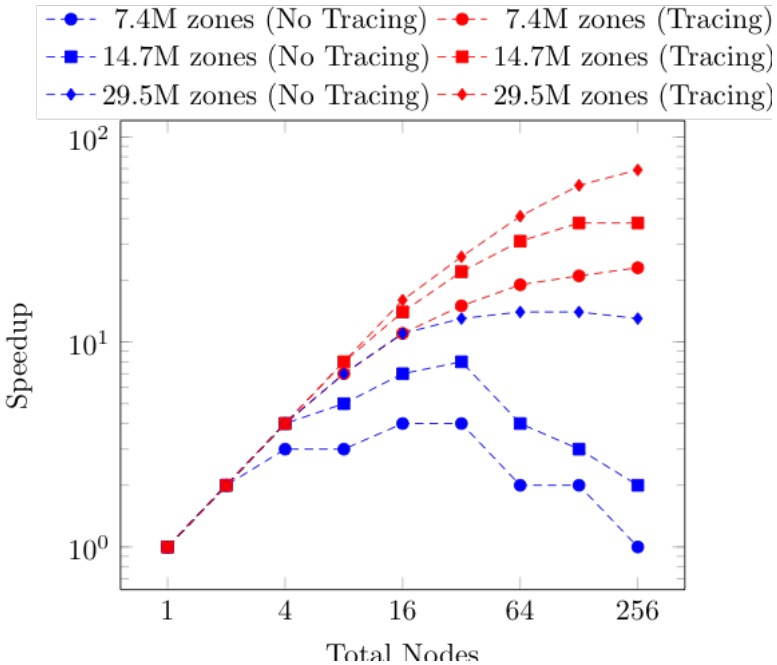


Figure 72: New Legion features such as Tracing will improve strong scaling in unstructured mesh computations.

programing 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 systesm.

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 Kokkos constructs directly into the parallel-IR representation. At present, this requires explicit lowering of Kokkos constructs and we have basic support for `parallel_for` and `parallel_reduce` in place. In addition we are looking at replacement of LLVM's dominator tree, a key data structure for optimizations including parallelization and memory usage analysis, with a *dominator directed-acyclic-graph* (DAG). This work is currently underway and should near its initial implementation early in the 2020 calendar year. In addition, 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. Furthermore, we are participating in ongoing discussions within the community about general details behind parallel intermediate forms.

Cinema: Recent Cinema work has focused on development of analysis capability, Exascale workflows and working with ECP STs such as ALPINE to enable in situ and post processing production of Cinema databases. Currently Cinema is available in situ via ParaView Catalyst and Ascent and available post hoc via ParaView and VisIt. New 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. In our most recent milestone, we ran two end-to-end simulation pipelines with ECP applications at scale to generate Cinema databases and ran Cinema-based workflows with Cinema algorithms to produce secondary set of artifacts. We ran (1) Nyx integrated with Ascent, running the ALPINE adaptive sampling algorithm; and (2) SW4 integrated with Ascent, running a VTK-m isocontour algorithm.

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 situp visualization and checkpoint-restart on a production system at LANL.

Next Steps

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 step for our feature set is to complete implementation of Kokkos use cases that match the needs of Ristra. Where possible, we will also explore the broader set of use cases within ECP's overall use of Kokkos constructs. The Kitsune toolchain is still very much an active *proof-of-concept* effort based on Clang (C and C++) with plans to add support for Fortran via the newly established Flang front end within LLVM (ST 2.3.2.12). Even though it is not yet production ready, we are actively updating and releasing source code and the supporting infrastructure for deployment as an early evaluation candidate. In addition to these components, we will actively begin to explore targeting the exascale systems (Aurora, Frontier, and El Capitan).

Cinema: Cinema is focusing on outreach to ECP applications to identify new application workflows that can be reasonably made efficient and working on new analysis methods for Cinema users.

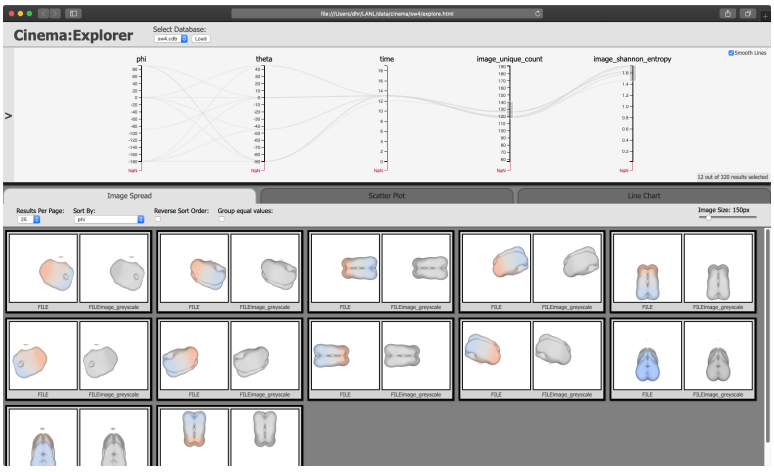


Figure 73: Screen capture of a browser-based viewer displaying the results of a analysis workflow using an SW4 isocontour Cinema database.

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 scientiic 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 [235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 2, 246, 247, 248, 249]. 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 [250] 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 [251], 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 3000 downloads/year from 70+ countries. It is freely available at `mfem.org`, on GitHub at `github.com/mfem`, where the MFEM community includes more than 165 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 project.

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.

The software products provided by this project are 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 Lawrence Livermore National Laboratory (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 [252, 253] is a next-generation resource management and scheduling software framework under active development at LLNL. This ECP project significantly augments the design and development of this framework to address two specific technical challenges pertaining to exascale computing.

1. Provide Flux as a portable user-level scheduling solution for complex exascale workflows
2. Provide capabilities for co-scheduling, high throughput, task coordination, and high portability.
3. Develop a resource model capable of portably representing job requirements of exascale systems.
4. Provide Flux as the system resource manager and scheduler for exascale systems.

Major efforts include developing and deploying additional capabilities such as management and scheduling of a diverse set of emerging workflows as well as a diverse set of exascale resources (e.g., power and burst buffers). The project strives to do this through co-design efforts with major workflow management software development teams within ASC (i.e., LLNL's UQPipeline), ECP/ATDM programs, and exascale computing hardware vendors themselves. Because Flux's design allows it to be used as a user-space scheduling tool, it is suitable for co-development with other workflow systems that require advanced scheduling capabilities. As a system tool, it is a potential replacement for resource managers such as SLURM, providing more advanced scheduling capabilities with full awareness of resources beyond just nodes and CPUs (e.g., filesystems, power, accelerators).

AID (Advanced Infrastructure for Debugging) provides an advanced debugging, code-correctness and testing toolset to facilitate reproducing, diagnosing and fixing bugs within HPC applications. The current capabilities include:

- 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 toolset 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

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:
Robust high-order finite element methods for ALE compressible flow. While high-order methods offer significant advantages in terms of HPC performance, their application to complicated ALE problems requires careful considerations to control oscillations and ensure accuracy.
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

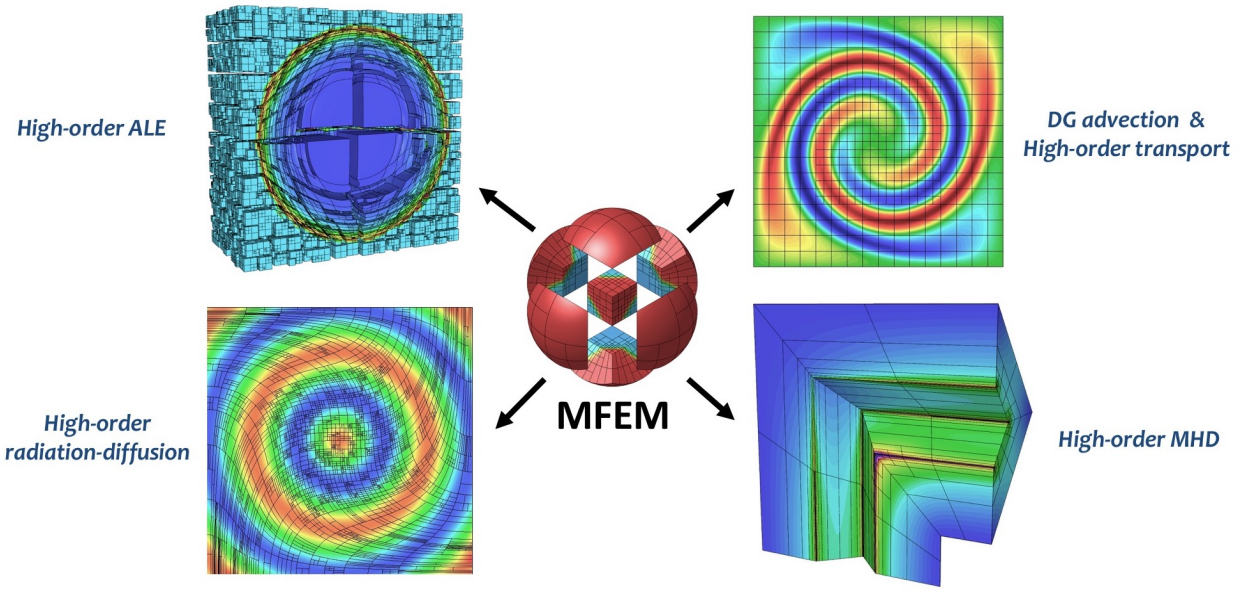


Figure 74: AMR implementation in MFEM allows many applications to benefit from non-conforming adaptivity, without significant changes in their codes.

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. Due 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/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 addres this by giving applicationst 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 performnance 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 applicaitons 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: Exascale resource management is particularly complex as it requires us to manage both the complexity of workloads (workflows, jobs, and services) as well as the increasing complexity of exascale machines themselves. Exascale systems may have diverse node types with CPUs, GPUs, burst buffers, and other independently allocatable hardware resources. Jobs must be mapped to these systems generically – one application must be able to run protably *and* with high performance or throughput on *any* exascale machine. Flux aims to save application developers the pain of configuring and setting up their applciations and workflows across multiple machine, and to enable massive ensembles and workflows to run scalably on

these machines.

AID: Debugging parallel applications running on supercomputers is extremely challenging. greater challenges. 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 assocciated with key application constructs that give them meaning. For example, we may know the number of floating point instructions over an entire applciation 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

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 [254, 255, 256, 257, 258, 259, 260, 261]. 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, derifenement 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 74.

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 increasingq 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 Office of Science, academia and industry, the MFEM team delivers regular releases on GitHub (e.g. mfem-3.3 in 2017, mfem-3.4 in 2018, mfem-4.0 in 2019) that include detailed documentation and many example codes. Code quality is ensured by smoke tests with Travis CI on Linux, Mac, Windows and nightly regression testing at LLNL.

RAJA/Umpire/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 performantly even if some functionality from OpenMP, CUDA, threading, etc. is missing.

Flux: Flux implements *hierarchical* scheduling. Ultimately, it will be usable either as a full system resource manager, *or* as a scheduler for a single workflow *within* another allocation, *or* as both. Flux allows application-level workloads to choose their own scheduleing policies and to specify concisely and portably the types of resources they need to run on a range of machines. Unlike prior approaches like SLURM, which use a on-size-fits-all scheduling and job management appoach, Flux allows the system to set global allocation policies, but users can instnatiate their own schedulers and request specific resources within an allocation. With Flux, users have the control over policy and scalability that was previously only tunable at the system level.

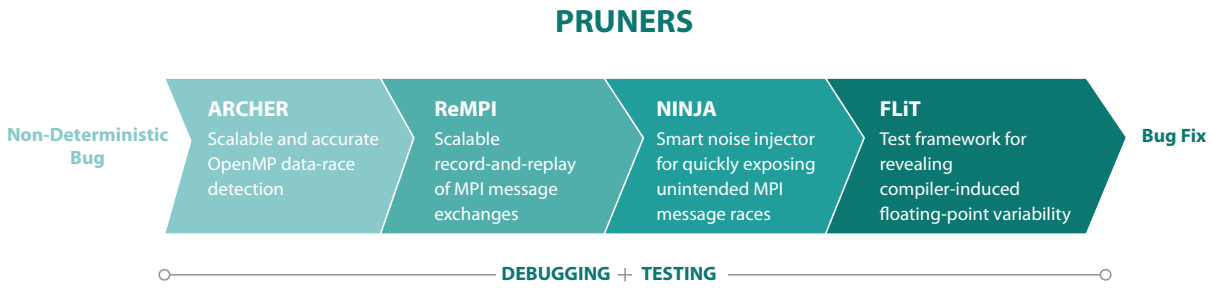


Figure 75: STAT, Archer, NINJA, and FliT: a continuum of debugging tools for exascale.

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.

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 automatiaclly 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 measuements and enable deeper analysis of the relationships between different parts of the code.

Recent Progress

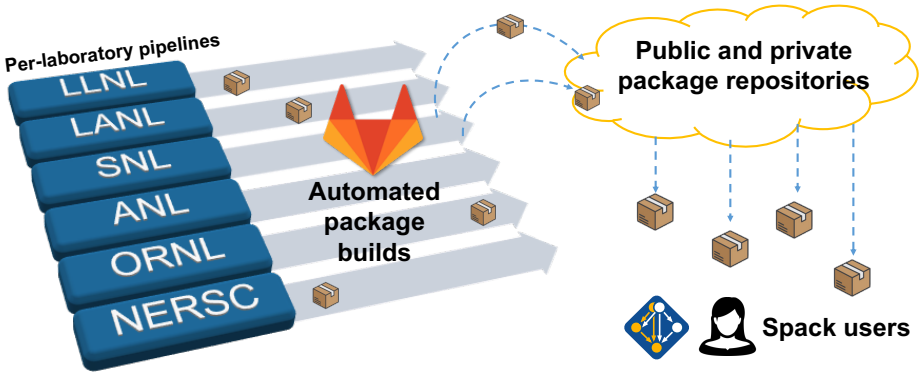


Figure 76: Spack build pipelines at facilities will provide HPC-native binary builds for users.

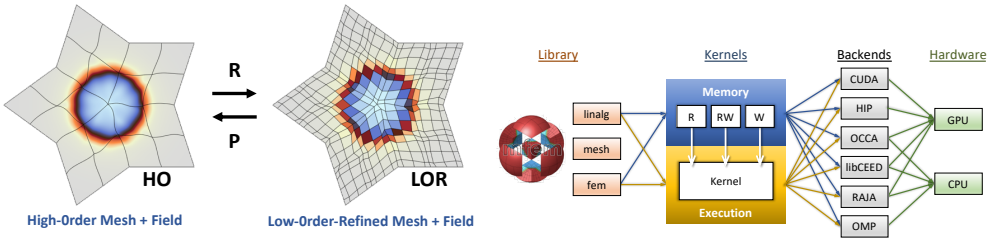


Figure 77: The MFEM team has developed High-Order ↔ Low-Order Transformations and GPU support for many linear algebra and finite element operations

Spack:

- Spack won a 2019 R&D 100 award as well as a Special Recognition as a Silver Medalist for being a "Market Disruptor".
- Completed the implementation of *Spack Stacks*: an extension of Spack Environments that enable large combinatorial facility deployments to be specified in a single file.
- Integrated Spack Environments and Spack Stacks with GitLab CI. This allows hundreds of builds to be farmed out to runners at facilities and in the cloud. Five organizations (NERSC, ANL, ORNL, NMC and the E4S team) were able to get pipelines working at their sites to automate builds.
- Implemented a new prototype *concretizer* for Spack. This version targets the NP-hard dependency resolution problem with an *Answer Set Programming* (ASP) based solver. Preliminary results show that solves of complex Spack stacks can be completed in seconds and that this tool can handle complex backtracking cases and optimization of package criteria that the existing greedy concretizer cannot.
- Spack has been selected as the package manager for Fugaku, Japan's flagship pre-exascale platform, and the team has been collaborating with RIKEN, Fujitsu, and SNL's Astra team to support the ARM platform.

MFEM: Selected recent highlights:

- Developed ALE discretization methods that support *completely lossless* high order to low order transformations, and vice versa.
- Delivered MFEM 4.0 release, with initial GPU support for many linear algebra and finite elementn operations.

- Developed a new formulation for discretizing problems with 1D spherical symmetry in BLAST. Extended BLAST to the 3T-model (separate equations for the electron and ion internal energies). Added support for changing masses during the Lagrangian phase.
- Completed the delivery of discretization support for the FY18 MARBL ATDM L2 milestone, including new 3T radiation-diffusion algorithms, and a simulation capability for problems with spherical and cylindrical symmetry via weight adjustments.
- Worked on high-order ALE algorithms in BLAST: developed remap step for density component masses, various code improvements and bugs fixes related to L2 milestone.

Machine	RAJA	CHAI	Umpire
Perlmutter	CUDA support production ready, actively used on Sierra. Continuing to investigate and improve performance		
Frontier	Support developed by AMD, PR pending. Will be released by end of CY19.	Available in CHAI v1.2.0 Developed by AMD	Available in Umpire v1.0.0 Developed by AMD
Aurora	Active area of research, also supported by ECP 2.3.1.18 RAJA/Kokkos. Programming model choice still under active development. SYCL is new area of research. OpenMP 4.5 in RAJA already developed for Sierra.		

Figure 78: Status of RAJA, Umpire, and CHAI support for exascale platforms.

RAJA/Umpire/CHAI: RAJA

- Comprehensive support for Sierra (Power9/Volta), including multi-dimensional kernel dispatch.
- enabled first full-system run on Sierra (16k GPUs, 97B elements)
- Added support for atomic operations on GPU devices.
- Integrated with GEOS, SW4, SUNDIALS, DevilRay, and LLNL ATDM.

Umpire

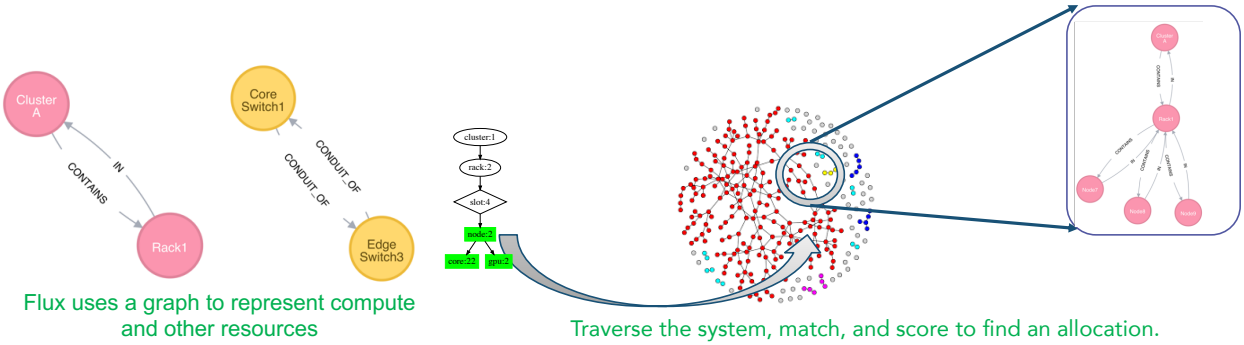
- Developed support for Sierra (Power9 + Volta) systems, incl. allocation on CPU, GPU, unified, and "pinned" memory resources; copying bt/w any resources; fast memory pools; CUDA "memory advice".
- Completed integration with multiple LLNL ASC applications and libraries, SW4, GEOS, and DevilRay. Began integration with LLNL ATDM application.

CHAI

- Developed Umpire-based backend for CHAI which adds additional flexibility and capability.
- Add option to pass specific Umpire objects (like pooled allocators) to CHAI arrays to improve application performance.
- Integrated with GEOS (ECP App Subsurface) application.

Flux:

- Extended graph-based resource model to support multi-user systems. This enables Flux to be used as a full-system resource manager, with security and isolation among usres
- Demonstrated end-to-end capability of DYAD data movement subsystem for LBANN.
- Completed definition of workflow exception/error handling model.



- Enabled two major scientific workflows to complete their calculations on LLNL's Sierra pre-exascale systems.
- Released flux-core 0.11 and flux-sched 0.7 that contain all of the functionalities used by these workflows.
- Started to broaden outreach and collaboration across ECP (e.g., scheduler integration working group), DOE complexes (e.g., ORNL, SNL and LANL), universities (e.g., UTK) and vendors (e.g., IBM T.J. Watson).

AID:

- Completed the port of STAT, Archer, FLiT, ReMPI/NINJA for Sierra, deployed them on these systems, and assisted users with these tools for debugging and testing.
- Isolated many elusive bugs for applications running on these systems, which includes large-scale code hangs due to NVIDIA GPU for a major ASC code.
- Archer and ReMPI have been integrated and/or tested with major ASC codes and have been running with their verification runs.
- Started to co-design and harden floating-point correctness checking tools (i.e., FLiT and FPChecker) with a large ASC code.

Caliper:

- Caliper has been integrated with ASC codes such as ARES and ARDRA.
- Caliper has been integrated with LLNL's SPOT performance tracking tool, which provides code teams with nightly performance data.

Next Steps

Spack: In FY20, the team will focus on:

- Enhancing Spack's dependency model to treat compilers as dependencies, so that we can better model ABI compatibility in our stacks.
- Parallel builds for Spack: enable Spack to run inside a SLURM allocation to efficiently install a large number of packages at once.
- Better detection and integration with external dependencies.
- Continued support of LLNL ATDM, other labs' ATDM teams, and facilities.

MFEM: The MFEM team will next demonstrate the use of our HO/LO mappings on general unstructured meshes in ATDM application at scale. This includes new discretization enhancements and new algorithms for ALE multi-physics applications, in particular in support of the MARBL appliaction's L2 milestone, especially with respect to the transition to exascale hardware. We will have MFEM running on early access machines as soon as they become available, and will support production runs on exascale platforms.

RAJA/Umpire/CHAI: Work in FY20–FY23 will focus on supporting El Capitan and other exascale-class systems available during this time frame. Additional work will support ASC and ATDM applications performance production runs on Sierra and integrating Umpire into additional LLNL WSC software components like Sidre, a simulation data store supported under the Axom project in the LLNL national security application project.

- Support LLNL ASC and ATDM applications with Sierra production runs
- Add El Capitan support to RAJA, CHAI, and Umpire
- Integrate Umpire with Sidre
- Support LLNL ASC and ATDM applications with transition to exascale systems

Flux:

- Deeper integration wtih Cancer Moonshot Pilot2 code, and LLNL ML initiative.
- Deeper LLNL UQ Pipeline integration.
- Testing of LLNL MARBL code with Flux on the SNL Astra system.

.

AID: In FY20–FY23, the gap analysis will be completed, and the team will closely work with the hardware vendors to fill these gaps for El Capitan and other systems.

Caliper: LLNL's ProTools continues to add Caliper into more and more ASC/ATDM codes so that all codes report their performance to a central dashboard. A large number of ATDM codes and libraries have asked for Caliper support, and the team's goal in the coming year is to satisfy all of these demands. The end result of this is to have application users running codes, producing behind-the-scenes performance data, and then application developers browsing and analyzing the performance data with analytic frameworks and novel visualizations.

4.6.7 WBS 2.3.6.03 SNL ATDM Software Technologies

Overview

The SNL 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

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 (2), 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 full 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 [262, 204]. 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 High Performance Computing 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

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. 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.

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 simultaneous 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 [204] *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

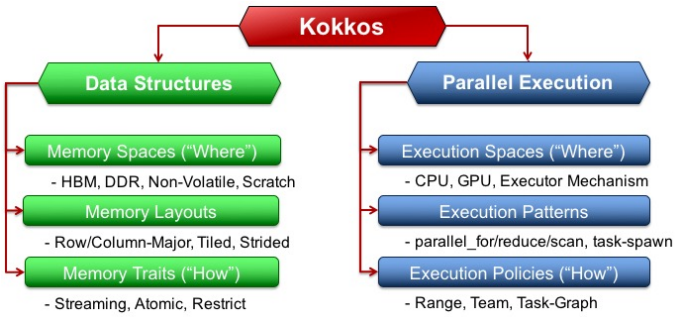


Figure 79: Kokkos Execution and Memory Abstractions

device adapter is constructed from a small but versatile set of data parallel primitives, which can be optimized for each platform [189]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [190, 191, 192].

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

Kokkos: Kokkos provided a production quality performance portability abstraction to applications and software technology projects under ATDM and ECP which allows them to run on all currently deployed DOE production compute platforms. Support for new platforms was generally in place on the relevant testbeds, before the production machines were delivered. The development of a number of new features based on customer needs improved the applicability of Kokkos for a wide range of applications. These include abstractions to seamlessly switch between data replication and atomic operation for scatter-add algorithms, improved flexibility of subview, tiled layouts, and multi-dimensional loop abstractions. Direct help with optimization work for applications, helped identify optimal algorithmic choices as well as improvements in the use of Kokkos.

Kokkos Kernels: Kokkos Kernels delivered performance portable kernels for the solve phase of the hypersonic simulations in SPARC. The default solver uses dense linear algebra kernels in Kokkos Kernels. This allows SPARC to achieve portable performance on CPU, Intel KNLs and GPUs. In addition, Kokkos Kernels are the default option for several linear algebra kernels in SNL EMPIRE simulations and the Kokkos Kernels symmetric Gauss-Seidel preconditioner, coloring algorithms are the default solver for the momentum equations in Exawind application.

VTK-m: VTK-m 1.4 was released in November 2018, with numerous features and improvements, including ZFP compression, clipping, connected components, particle advection and others. See the ECP VTK-m report for details. In addition, the SPARC/Catalyst interface was updated to address some changes to the SPARC code including a change of the SPARC parser to a yaml format for its input. The team also Integrated and hardened the SPARC/Catalyst interface to make it feasible to merge into the SPARC master branch. The SPARC/Catalyst capability enables in situ capabilities to extract surface data of a re-entry vehicle at scale. The team also developed a prototype of functional tensor approximation/compression using

subsets/slices of Openjet data. These include interpolation onto structured mesh followed by JPEG-like compression, Tucker compression, canonical low-rank functional approximation and functional tensor-train.

OS & ONR: The OS & ONR team had a number of recent accomplishments. The KVM hypervisor was enabled on a Cray XC30 system and application performance studies were conducted to determine the overhead of running in a virtual machine versus the native OS. This is the first demonstration of a virtual cluster on a Cray system. They enabled the use of the Singularity container system on a Cray system and compared the performance of identical containers running on Cray and Amazon EC2 hardware. They enabled the effective coordination of on-node resources between multiple OS/R environments to evaluate and improve performance isolation capabilities. They extended the LogGOPSim simulator to track MPI resource usage without perturbing applications in order to better understand the impact of MPI matching behavior to guide hardware implementation choices. They performed a scaling study comparing a containerized version of Nalu with a native version on a CTS-1 platform, which demonstrated that the container can actually reduce runtime while consuming more memory. Made contributions to the ongoing activities of the MPI Forum and the OpenMP Language Committee. Key efforts include: MPI persistent collectives, MPI sessions, MPI finepoints, OpenMP runtime interoperability, and OpenMP multi-level memory management.

Next Steps

Kokkos: The Kokkos team is working with the Path Forward vendors to enable support for their architectures. This notably includes a new backend, called ROCm, for AMD GPUs, which is primarily developed by AMD itself. Furthermore, improvements on the dynamic task graph execution on GPUs are planned, in order to reduce the task granularity necessary to make effective use for GPUs.

Kokkos Kernels: The primary goal in FY20 is to achieve the performance requirements needed to complete the goals of the ATDM L1 milestone. This will focus on improving the performance of the key kernels used by SPARC and EMPIRE. The primary performance goal for supporting SPARC simulations is to improve the kernel efficiency on Volta architecture and MPI communication performance for the solvers on Sierra platform. The primary performance goal for supporting EMPIRE simulations is to adapt linear algebra kernels to the Volta architecture and achieve the simulation goals of EMPIRE in terms of number of linear solves per second.

VTK-m: Starting in FY20, the ATDM/VTK-m project will shift from primarily building functionality into the VTK-m toolkit to addressing the needs of other ST projects and ATDM applications. This work will focus on the three key goals of ECP: performance, integration, and quality. We are also exploring the feasibility of using Kokkos libraries to implement the device porting layer.

OS & ONR: Demonstrate containers on ATS-1 and ATS-2 to support ATDM developer workflows and demonstrate ATDM workloads running on vendor exascale OS/R stacks, evaluate performance and characterize impact of Sandia tech transfer. Refinement of node resource management and runtime with tech transfer to Kokkos and OpenMP and optimize OS/R resource usage for ATDM workloads and demonstrate performance impact. Contribute to the MPI and OpenMP specifications and engage vendors in support of MPI and OpenMP to meet the needs of Kokkos and ATDM apps.

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 Exascale Computing Project 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] 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.

[4] 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.

[5] LLVM Compiler Infrastructure. LLVM Compiler Infrastructure. http://www.llvm.org.

[6] Supercontainers Presentation. https://oaciss.uoregon.edu/E4S-Forum19/talks/Younge-E4S.pdf.

[7] 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.

[8] Livermore Computing. Toss: Speeding up commodity cluster computing. https://computation.llnl.gov/projects/toss-speeding-commodity-cluster-computing.

[9] OpenHPC. Community building blocks for hpc systems. http://openhpc.community.

[10] 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.

[11] Rajeev Thakur and William Gropp. Test suite for evaluating performance of multithreaded MPI communication. *Parallel Comput.*, 35(12):608–617, December 2009.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Y. Guo, C. J. Archer, M. Blocksome, S. Parker, W. Bland, K. Raffenetti, 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.

[17] 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.

[18] Ken Raffenetti, 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.

[19] 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.

[20] 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.

[21] F. Ji, J. S. Dinan, D. T. Buntinas, P. Balaji, X. Ma and W. chun Feng. Optimizing GPU-to-GPU intra-node communication in MPI. In *2012 International Workshop on Accelerators and Hybrid Exascale Systems, AsHES 12*, 2012.

[22] Lena Oden and Pavan Balaji. Hexe: A toolkit for heterogeneous memory management. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2017.

[23] Giuseppe Congiu and Pavan Balaji. Evaluating the impact of high-bandwidth memory on mpi communications. In *IEEE International Conference on Computer and Communications*, 2018.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] Abdelhalim Amer, Charles Archer, Michael Blocksome, Chongxiao Cao, Michael Chuvelev, Hajime Fujita, Maria Garzaran, Yanfei Guo, Jeff R. Hammond, Shintaro Iwasaki, Kenneth J. Raffenetti, 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.

[32] 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.

[33] Michael Bauer and Michael Garland. Legate numpy: Accelerated and distributed array computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '19, page to appear, 2019.

[34] Elliott Slaughter and Alex Aiken. Pygion: Flexible, scalable task-based parallelism with python. In *Proceedings of Parallel Applications Workshop, Alternatives To MPI+X, co-located with SC19*, page to appear, 2019.

[35] Zhihao Jia, Matei Zaharia, and Alex Aiken. Beyond data and model parallelism for deep neural networks. *CoRR*, abs/1807.05358, 2018.

[36] Wonchan Lee, Elliott Slaughter, Michael Bauer, Sean Treichler, Todd Warszawski, Michael Garland, and Alex Aiken. Dynamic tracing: Memoization of task graphs for dynamic task-based runtimes. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, pages 34:1–34:13, Piscataway, NJ, USA, 2018. IEEE Press.

[37] 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.

[38] 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.

[39] Thomas Herault, Yves Robert, George Bosilca, and Jack J. Dongarra. Generic matrix multiplication for multi-GPU accelerated distributed-memory platforms over PaRSEC. Scala19, 2019.

[40] GASNet website. https://gasnet.lbl.gov/.

[41] John Bachan, Dan Bonachea, Paul H. Hargrove, Steve Hofmeyr, Mathias Jacquelin, Amir Kamil, Brian van Straalen, and Scott B. Baden. The UPC++ PGAS library for exascale computing. In *Proceedings of the Second Annual PGAS Applications Workshop*, PAW17, pages 7:1–7:4, New York, NY, USA, 2017. ACM. https://doi.org/10.1145/3144779.3169108.

[42] UPC++ website. https://upcxx.lbl.gov/.

[43] 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.

[44] The Legion Programming System website. http://legion.stanford.edu/.

[45] Bradford L. Chamberlain. Chapel. In *Programming Models for Parallel Computing*. The MIT Press, 2015.

[46] The Chapel Parallel Programming Language website. https://chapel-lang.org/.

[47] 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.

[48] 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.

[49] Paul H. Hargrove and Dan Bonachea. Efficient Active Message RMA in GASNet using a target-side reassembly protocol (extended abstract). Technical Report LBNL-2001238, Lawrence Berkeley National Laboratory, November 2019. To appear: 2019 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM). https://doi.org/10.25344/S4PC7M.

[50] Paul H. Hargrove and Dan Bonachea. GASNet-EX performance improvements due to specialization for the Cray Aries network. Technical Report LBNL-2001134, Lawrence Berkeley National Laboratory, March 2018. https://doi.org/10.2172/1430690.

[51] John Bachan, Scott B. Baden, Dan Bonachea, Paul H. Hargrove, Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, and Brian van Straalen. UPC++ v1.0 Specification, Revision 2019.9.0. Technical Report LBNL-2001237, Lawrence Berkeley National Laboratory, September 2019. https://doi.org/10.25344/S4ZW2C.

[52] 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.

[53] 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.

[54] RAJA Performance Portability Layer. https://github.com/LLNL/RAJA.

[55] CHAI Copy-hiding Array Abstraction. https://github.com/LLNL/CHAI.

[56] Umpire Application-focused Memory Management API. https://github.com/LLNL/Umpire.

[57] RAJA Performance Suite. https://github.com/LLNL/RAJAPerf.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] Johannes Doerfert and Hal Finkel. Compiler optimizations for openmp. In *International Workshop on OpenMP*, pages 113–127. Springer, 2018.

[64] Johannes Doerfert, Jose Manuel Monsalve Diaz, and Hal Finkel. The tregion interface and compiler optimizations for openmp target regions. In *International Workshop on OpenMP*, pages 153–167. Springer, 2019.

[65] Michael Kruse and Hal Finkel. User-directed loop-transformations in clang. In *2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, pages 49–58. IEEE, 2018.

[66] Michael Kruse and Hal Finkel. Loop optimization framework. *LCPC 2018; arXiv:1811.00632*, 2018.

[67] Hal Finkel, David Poliakoff, and David F Richards. Clangjit: Enhancing c++ with just-in-time compilation. *arXiv preprint arXiv:1904.08555*, 2019.

[68] Hal Finkel. P1609r0: C++ should support just-in-time compilation. http://wg21.link/p1609r0, 2019.

[69] OpenACC: Commerical Compilers. [Online]. Available: http://openacc.org/tools.

[70] 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.

[71] SPEC ACCEL. [Online]. Available: https://www.spec.org/accel/.

[72] 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.

[73] 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.

[74] T. Zhao, S. Williams, M. Hall, and H. Johansen. Pack-free stencil ghost zone exchange. In *submitted to IPDPS '20*, 2019.

[75] Evangelos Georganas, Aydin Buluç, Jarrod Chapman, Leonid Oliker, Daniel Rokhsar, and Katherine Yelick. Parallel De Bruijn Graph Construction and Traversal for De Novo Genome Assembly. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 437–448, 2014.

[76] 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].

[77] 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.

[78] 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.

[79] Johannes Doerfert, Jose Diaz, and Hal Finkel. *The TRegion Interface and Compiler Optimizations for OpenMP Target Regions*, pages 153–167. 08 2019.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] Seonmyeong Bak, Yanfei Guo, Pavan Balaji, and Vivek Sarkar. Optimized Execution of Parallel Loops via User-Defined Scheduling Policies. In *ICPP*, 2019.

[88] 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.

[89] 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.

[90] Ralf S. Engelschall. GNU portable threads (Pth). http://www.gnu.org/software/pth, 1999.

[91] 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.

[92] S. Thibault. A flexible thread scheduler for hierarchical multiprocessor machines. In *COSET*, 2005.

[93] J. Nakashima and Kenjiro Taura. MassiveThreads: A thread library for high productivity languages. In *Concurrent Objects and Beyond*, pages 222–238. Springer, 2014.

[94] K. B. Wheeler, Richard C. Murphy, and Douglas Thain. Qthreads: An API for programming with millions of lightweight threads. In *MTAAP*, 2008.

[95] K. Taura, Kunio Tabata, and Akinori Yonezawa. StackThreads/MP: Integrating futures into calling standards. In *PPPoP*, pages 60–71, 1999.

[96] 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.

[97] Chuck Pheatt. Intel® threading building blocks. *Journal of Computing Sciences in Colleges*, 23(4):298–298, 2008.

[98] M. Pérache, Hervé Jourdren, and Raymond Namyst. MPC: A unified parallel runtime for clusters of NUMA machines. In *EuroPar*, pages 78–88, 2008.

[99] A. Adya, Jon Howell, Marvin Theimer, William J. Bolosky, and John R. Douceur. Cooperative task management without manual stack management. In *ATC*, 2002.

[100] CORPORATE SunSoft. *Solaris multithreaded programming guide*. Prentice-Hall, Inc., 1995.

[101] 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.

[102] G. Shekhtman and Mike Abbott. State threads library for internet applications. http://state-threads.sourceforge.net/, 2009.

[103] 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.

[104] A. Porterfield, Nassib Nassar, and Rob Fowler. Multi-threaded library for many-core systems. In *MTAAP*, 2009.

[105] 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.

[106] Intel OpenMP runtime library. https://www.openmprtl.org/, 2016.

[107] Barcelona Supercomputing Center. Nanos++. https://pm.bsc.es/projects/nanox/, 2016.

[108] L. V. Kalé, Josh Yelon, and T. Knuff. Threads for interoperable parallel programming. In *LCPC*, pages 534–552, 1996.

[109] S. Treichler, Michael Bauer, and Alex Aiken. Realm: An event-based low-level runtime for distributed memory architectures. In *PACT*, pages 263–276, 2014.

[110] 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.

[111] 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.

[112] 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.

[113] 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.

[114] 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.

[115] Fortran Standards Committee. Information technology – Programming languages – Fortran – Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, October 2004.

[116] Fortran Standards Committee. Information technology – Programming languages – Fortran – Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, October 2010.

[117] Fortran Standards Committee. Draft International Standard – Information technology – Programming languages – Fortran – Part 1: Base language. Standard, International Organization for Standardization, Geneva, CH, December 2017.

[118] LLVM Project Team. LLVM Web page. https://llvm.org.

[119] 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, 2020.

[120] xSDK Web page. http://xsdk.info.

[121] xSDK Community Policies Web page. http://xsdk.info/policies.

[122] xSDK developers. xSDK community package policies, 2019. version 0.5.0, June 2019, https://dx.doi.org/10.6084/m9.figshare.4495136.

[123] xSDK developers. xSDK community installation policies: GNU Autoconf and CMake options, 2019. version 0.5.0, June, 2019, https://dx.doi.org/10.6084/m9.figshare.4495133.

[124] hypre Web page. https://computation.llnl.gov/projects/hypre.

[125] PETSc/TAO Team. PETSc/TAO website. https://www.mcs.anl.gov/petsc.

[126] SuperLU Web page. http://crd-legacy.lbl.gov/~xiaoye/SuperLU.

[127] Trilinos Web page. https://trilinos.org.

[128] MAGMA Web page. http://icl.utk.edu/magma.

[129] MFEM Web page. http://mfem.org.

[130] SUNDIALS Web page. https://computation.llnl.gov/projects/sundials.

[131] AMReX Web page. https://amrex-codes.github.io/amrex/.

[132] DTK Web page. https://github.com/ORNL-CEES/DataTransferKit.

[133] Omega_h Web page. https://github.com/ibaned/omega_h.

[134] PLASMA Web page. https://bitbucket.org/icl/plasma.

[135] PUMI Web page. https://github.com/SCOREC/core.

[136] STRUMPACK Web page. http://portal.nersc.gov/project/sparse/strumpack/.

[137] Tasmanian Web page. https://tasmanian.ornl.gov.

[138] deal.ii Web page. https://www.dealii.org/.

[139] PHIST Web page. https://bitbucket.org/essex/phist.

[140] SLEPc Web page. http://slepc.upv.es/.

[141] ButterflyPACK Web page. https://github.com/liuyangzhuan/ButterflyPACK.

[142] Ginkgo Web page. https://github.com/ginkgo-project/ginkgo.

[143] libEnsemble Web page. https://github.com/Libensemble/libensemble.

[144] preCICE Web page. https://www.precice.org.

[145] Alquimia Web page. https://bitbucket.org/berkeleylab/alquimia.

[146] PFLOTRAN Web page. http://www.pflotran.org.

[147] 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 3.10. Technical Memorandum ANL-95/11 – Revision 3.10, Argonne National Laboratory, 2018.

[148] 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.

[149] V. Henson and U. Yang. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41:155–177, 2002.

[150] 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.

[151] 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.

[152] 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.

[153] 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.

[154] 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.

[155] S. Tomov, J. Dongarra, and M. Baboulin. Towards dense linear algebra for hybrid gpu accelerated many-core systems. *Parellel Comput. Syst. Appl.*, 36(5-6):232–240, 2010. DOI: 10.1016/j.parco.2009.12.005.

[156] 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.

[157] 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.

[158] 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.

[159] I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra. Improving the Performance of CA-GMRES on Multicores with Multiple GPUs. In *28th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2014)*, 2014.

[160] Jeffrey Cornelis, Siegfried Cools, and Wim Vanroose. The communication-hiding conjugate gradient method with deep pipelines. *CoRR*, abs/1801.04728, 2018.

[161] E. Chow, H. Anzt, and J. Dongarra. Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs. In *Lecture Notes in Computer Science*, volume 9137, pages 1–16, July 12 – 16 2015.

[162] H. Anzt, E. Chow, and J. Dongarra. Iterative sparse triangular solves for preconditioning. In Jesper Larsson Träff, Sascha Hunold, and Francesco Versaci, editors, *Euro-Par 2015: Parallel Processing*, volume 9233 of *Lecture Notes in Computer Science*, pages 650–661. Springer Berlin Heidelberg, 2015.

[163] Hartwig Anzt, Thomas K. Huckle, Jürgen Bräckle, and Jack Dongarra. Incomplete sparse approximate inverses for parallel preconditioning. *Parallel Computing*, 71(Supplement C):1 – 22, 2018.

[164] Erin Carson and Nicholas J. Higham. A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems. *SIAM J. Scientific Computing*, 39(6):A2834–A2856, 2017.

[165] Erin Carson and Nicholas J. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Scientific Computing*, 40(2):A817–A847, 2018.

[166] Hartwig Anzt, Jack Dongarra, Goran Flegar, Nicholas J. Higham, and Enrique S. Quintana-Ortí. Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers. *Concurrency and Computation: Practice and Experience*, 31(6):e4460, 2019. e4460 cpe.4460.

[167] Better Scientific Software (BSSw) https://bssw.io/.

[168] 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.

[169] I. Yamazaki, S. Thomas, M. Hoemmen, Erik G. Boman, and K. Swirydowicz. Low-synchronization orthogonalization for s-step and pipelined krylov solvers in Trilinos. In *Proc. of SIAM Parallel Processing 2020*, 2020. submitted.

[170] H. Anzt, E. Chow, and J. Dongarra. ParILUT—A New Parallel Threshold ILU Factorization. *SIAM Journal on Scientific Computing*, 40(4):C503–C519, 2018.

[171] H. Anzt, T. Ribizel, G. Flegar, E. Chow, and J. Dongarra. Parilut - a parallel threshold ilu for gpus. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 231–241, May 2019.

[172] Goran Flegar, Hartwig Anzt, Terry Cojean, and Enrique S. Quintana-Ortí. Customized-Precision Block-Jacobi Preconditioning for Krylov Iterative Solvers on Data-Parallel Manycore Processors. *ACM Trans. on Mathematical Software*, submitted.

[173] Heike Jagode, Anthony Danalis, Hartwig Anzt, and Jack Dongarra. Papi software-defined events for in-depth performance analysis. *The International Journal of High Performance Computing Applications*, 0(0):1094342019846287, 0.

[174] Ginkgo Performance Explorer. https://ginkgo-project.github.io/gpe/. 2019.

[175] 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.

[176] 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.

[177] 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.

[178] David M Beazley et al. SWIG: An easy to use tool for integrating scripting languages with C and C++. In *4th Conference on USENIX Tcl/Tk Workshop*, 1996.

[179] 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.

[180] ADIOS2 documentation. http://adios2-adaptable-io-system-version-2.readthedocs.io.

[181] The ADIOS2 framework. https://github.com/ornladios/ADIOS2.

[182] 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.

[183] 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.

[184] Kenneth Moreland. The ParaView tutorial, version 4.4. Technical Report SAND2015-7813 TR, Sandia National Laboratories, 2015.

[185] 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.

[186] 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.

[187] 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.

[188] 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.

[189] Guy E. Blelloch. *Vector Models for Data-Parallel Computing*. MIT Press, 1990. ISBN 0-262-02313-X.

[190] 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.

[191] 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.

[192] 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.

[193] 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.

[194] 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.

[195] Peter Lindstrom. Error distributions of lossy floating-point compressors. In *JSM 2017 Proceedings*, pages 2574–2589, 2017.

[196] Abhishek Yenpure, Hank Childs, and Kenneth Moreland. Efficient point merging using data parallel techniques. In *Eurographics Symposium on Parallel Graphics and VIsualization (EGPGV)*, June 2019.

[197] David Pugmire, Abhishek Yenpure, Mark Kim, James Kress, Robert Maynard, Hank Childs, and Bernd Hentschel. Performance-portable particle advection with VTK-m. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 45–55, June 2018.

[198] Bogdan Nicolae, Adam Moody, Elsa Gonsiorowski, Kathryn Mohror, and Franck Cappello. Veloc: Towards high performance adaptive asynchronous checkpointing at large scale. In *IPDPS'19: The 2019 IEEE International Parallel and Distributed Processing Symposium*, pages 911–920, Rio de Janeiro, Brazil, 2019.

[199] Shu-Mei Tseng, Bogdan Nicolae, George Bosilca, Emmanuel Jeannot, Aparna Chandramowlishwaran, and Franck Cappello. Towards portable online prediction of network utilization using mpi-level monitoring. In *EuroPar'19 : 25th International European Conference on Parallel and Distributed Systems*, pages 1–14, Goettingen, Germany, 2019.

[200] Utkarsh Ayachit. The ParaView guide: a parallel visualization application, 2015.

[201] James Ahrens, Berk Geveci, and Charles Law. ParaView: An end-user tool for large-data visualization. *The visualization handbook*, 2005.

[202] 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.

[203] 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.

[204] 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.

[205] Brad Whitlock, Jean Favre, and Jeremy Meredith. Parallel in situ coupling of simulation with a fully featured visualization system. In *Proceedings of the 11th Eurographics Conference on Parallel Graphics and Visualization (EGPGV)*, 2011.

[206] 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.

[207] 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.

[208] 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.

[209] 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.

[210] A Biswas, S Dutta, J Pulido, and J Ahrens. In situ data-driven adaptive sampling for large-scale simulation data summarization. *In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization, ISAV*, pages 13–18, 2018.

[211] Soumya Dutta, Ayan Biswas, and James Ahrens. Multivariate pointwise information-driven data sampling and visualization. *Entropy*, 21(7):1–25, 2019.

[212] 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.

[213] Roxana Bujack and Jan Flusser. Flexible moment invariant bases for 2d scalar and vector fields. In *Proceedings of International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 11–20, 2017.

[214] Bo Yang, Jitka Kostkova, Jan Flusser, Tomas Suk, and Roxana Bujack. Rotation Invariants of Vector Fields from Orthogonal Moments. *Pattern Recognition*, pages 110–121, 2018.

[215] Bei Wang, Roxana Bujack, Harsh Bhatia Paul Rosen, Primoz Skraba, and Hans Hagen. Interpreting galilean invariant vector field analysis via extended robustness. In *Topology-Based Methods in Visualization (TopoInVis 2017) Tokyo, Japan*, 2017.

[216] S. Petruzza, S. Treichler, V. Pascucci, and P. 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, May 2018.

[217] 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.

[218] 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.

[219] 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.

[220] Franck Cappello and Peter Lindstrom. Compression of scientific data. ISC High Performance 2017 Tutorials, 2017.

[221] Franck Cappello and Peter Lindstrom. Compression of scientific data. IEEE/ACM SC 2017 Tutorials, 2017.

[222] Franck Cappello and Peter Lindstrom. Compression for scientific data. Euro-Par 2018 Tutorials, 2018.

[223] Franck Cappello and Peter Lindstrom. Compression for scientific data. IEEE/ACM SC 2018 Tutorials, 2018.

[224] Franck Cappello and Peter Lindstrom. Compression for scientific data. ISC High Performance 2019 Tutorials, 2019.

[225] Franck Cappello, Peter Lindstrom, and Sheng Di. Compression for scientific data. IEEE/ACM SC 2019 Tutorials, 2019.

[226] Peter Lindstrom, Markus Salasoo, Matt Larsen, and Stephen Herbein. ZFP version 0.5.5, May 2019. https://github.com/LLNL/zfp.

[227] E4S Web page. http://e4s.io.

[228] 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.

[229] Docker Inc. Docker. https://www.docker.com.

[230] BEE. BEE. http://bee.dsscale.org.

[231] RS Canon and D Jacobsen. Shifter: Containers for hpc. In *Proceedings of the Cray User's Group*, 2016.

[232] 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.

[233] 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.

[234] 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.

[235] Adam J. Stewart, Massimiliano Culpo, Gregory Becker, Peter Scheibel, and Todd Gamblin. Spack Community BoF. In *Supercomputing 2019*, Denver, CO, November 21 2019.

[236] 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.

[237] Todd Gamblin and Gregory Becker. Tutorial: Spack for Developers. In *Los Alamos National Laboratory*, Los Alamos, NM, November 5 2019. Full day.

[238] 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.

[239] 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.

[240] 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.

[241] Todd Gamblin, Gregory Becker, Matthew P. LeGendre, and Peter Scheibel. Spack Roundtable Discussion. In *Exascale Computing Project 3nd Annual Meeting*, Houston, TX, January 16 2019.

[242] Todd Gamblin, Gregory Becker, Peter Scheibel, Matt Legendre, and Mario Melara. Managing HPC Software Complexity with Spack. In *Exascale Computing Project 3nd Annual Meeting*, Houston, TX, January 14 2019. Full day.

[243] 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.

[244] 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.

[245] 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.

[246] 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.

[247] Todd Gamblin. Tutorial: Managing HPC Software Complexity with Spack. In *HPC Knowledge Meeting (HPCKP'17)*, San Sebastián, Spain, June 16 2017. 2 hours.

[248] Gregory Becker, Matt Legendre, and Todd Gamblin. *Tutorial: Spack for HPC*. Livermore Computing, Lawrence Livermore National Laboratory, Livermore, CA, April 6 2017. Half day.

[249] 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.

[250] MFEM: Modular finite element methods library. mfem.org.

[251] BLAST: High-order finite element Lagrangian hydrocode. https://computation.llnl.gov/projects/blast.

[252] Dong H. Ahn, Jim Garlick, Mark Grondona, Don Lipari, Becky Springmeyer, and Martin Schulz. Flux: A next-generation resource management framework for Large hpc centers. In *Proceedings of the 10th International Workshop on Scheduling and Resource Management for Parallel and Distributed Systems*, September 2014.

[253] Dong H. Ahn, Ned Bass, Albert Chu, Jim Garlick, Mark Grondona, Stephen Herbein, Joseph Koning, Tapasya Patki, Thomas R. W. Scogland, and Becky Springmeyer. Flux: Overcoming scheduling challenges for exascale workflows. In *Proceedings of the 13th Workshop on Workflows in Support of Large-Scale Science*, WORKS '18, 2018.

[254] 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.

[255] 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.

[256] 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.

[257] 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.

[258] 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.

[259] 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.

[260] 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.

[261] 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.

[262] James Ahrens, Berk Geveci, and Charles Law. ParaView: An end-user tool for large data visualization. In *Visualization Handbook*. Elesvier, 2005. ISBN 978-0123875822.