ECP-U-AD-RPT-2022-00271

Application Results on Early Exascale Hardware

WBS 2.2, Milestone PM-AD-1140

Andrew Siegel[1], Erik W. Draeger[2], Jack Deslippe[3], Tom Evans[4], Marianne M. Francois[5], Tim Germann[5], Dan Martin[3], and William Hart[6]

[1]Argonne National Laboratory
[2]Lawrence Livermore National Laboratory
[3]Lawrence Berkeley National Laboratory
[4]Oak Ridge National Laboratory
[5]Los Alamos National Laboratory
[6]Sandia National Laboratories

March 31, 2022

For public release

ECP-U-AD-RPT-2022-00271

ECP Milestone Report
Application Results on Early Exascale Hardware
WBS 2.2, Milestone PM-AD-1140

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

March 31, 2022

ECP Milestone Report
Application Results on Early Exascale Hardware
WBS 2.2, Milestone PM-AD-1140

APPROVALS

Submitted by:

Andrew Siegel, Argonne National Laboratory PM-AD-1140	Date

Concurrence:

Doug Kothe Oak Ridge National Laboratory	Date

Approval:

Doug Kothe Oak Ridge National Laboratory	Date

REVISION LOG

Version	Creation Date	Description	Approval Date
1.0	August 1, 2022	Original	

EXECUTIVE SUMMARY

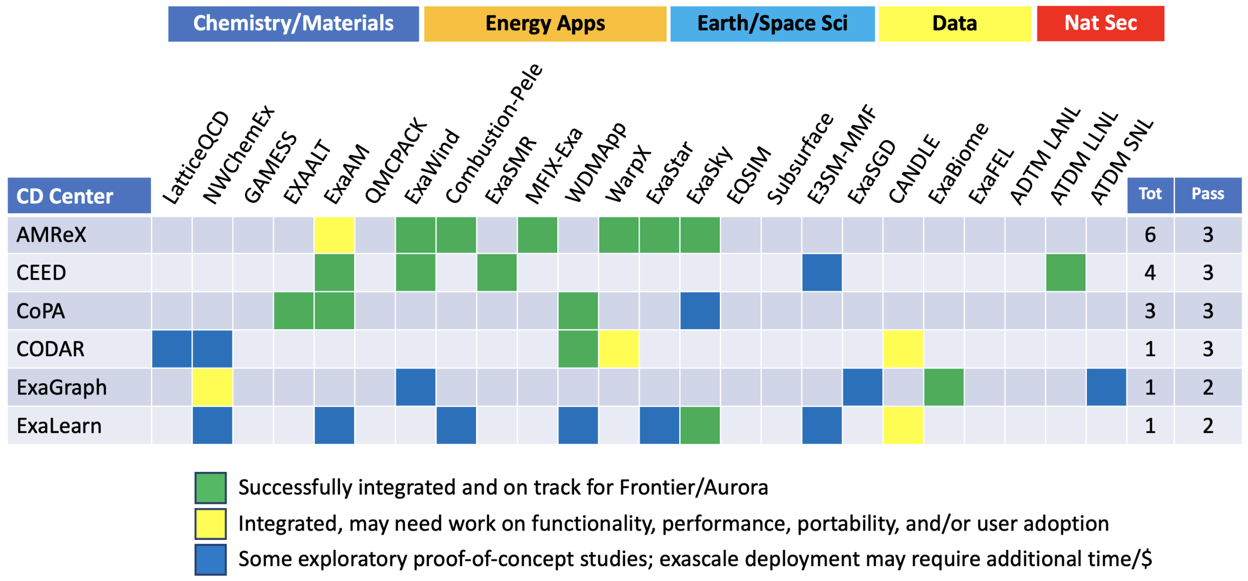
This Exascale Computing Project (ECP) milestone report summarizes the status of 27 of the 31 ECP Applications Development (AD) subprojects at the end of FY21. In November and December of 2021, a comprehensive assessment of AD projects was conducted by the ECP leadership along with external subject matter experts (SMEs). (NNSA application projects are reviewed separately using the ASC milestone process.) The AD review committee—consisting of the AD lead, AD deputy, Level 3 (L3), and at least one external project SME—was tasked with evaluating each project's progress relative to ECP project goals specified in the FY21 timeline. Key areas of focus were code maturity and performance on pre-exascale systems, an in-depth analysis of final key performance parameter (KPP) verification contracts, and future R&D priorities in the final year of ECP and beyond. As such, this report contains not only an accurate snapshot of each subproject's current status but also represents a broad account of successes and challenges in porting large scientific applications to DOE's next-generation high-performance computing architectures – the Frontier and Aurora systems.

One goal of a detailed annual assessment is to provide a global risk assessment – a clear, up to date snapshot of the likelihood of success of each AD subproject. All subprojects have made significant progress toward meeting their contribution to project KPPs. However, technical challenges remain, and for some projects the challenges are more significant than others. To capture risk at this level of granularity, we carried out a best-judgment analysis for each subproject, binning their probability of success as *near certainty*, *very likely*, *likely*, or *tossup*. Fifteen of the 21 KPP-1 and KPP-2 projects were scored as either near certainty or highly likely, with either no obstacles to success, or barriers that are expected to have solutions forthcoming. For six others the obstacles have been identified, but solutions, while expected, are longer-term and less well defined.

For KPP-3 applications, a tentative score was computed based on the current number of successful mature integrations. Additional integrations are scored as either maturing or exploratory. Based on this analysis, three of the six co-design projects are expected to meet their KPP-3 goal, with three others continuing to mature additional points of integration needed to meet their threshold. For KPP-1 and KPP-2 applications, key obstacles are listed in the table, and for all 27 projects greater detail is provided in body of the report.

Table 1: Overview of current risk assessment, showing the likelihood of each AD subproject meeting its KPP objective along with main obstacles to success.

AD Project	Application Area	KPP Projection	Key Challenges
EXAALT	Atomistic materials	Near Certainty	No Blockers
ExaSky	Cosmology	Near Certainty	No Blockers
ExaSMR	Nuclear energy	Very Likely	Portability
CANDLE	Precision medicine for oncology	Near Certainty	No Blockers
E3SM-MMF	Earth system science	Very Likely	Portability
EQSIM	Earthquakes	Near Certainty	No Blockers
LatticeQCD	Nuclear physics	Likely	Network Performance
NWChemEx	Chemistry	Near Certainty	No Blockers
QMCPACK	Quantum materials	Likely	OpenMP
WarpX	Accelerator physics	Near Certainty	No Blockers
WDMApp	Magnetic fusion energy	Very Likely	Algorithms,Optimization
Combustion-PELE	Combustion science	Very Likely	Sparse Solvers
ExaAM	Advanced manufacturing	Likely	Portability
ExaBiome	Microbiome analysis	Likely	UPC Support
ExaFEL	Laser analytics	Very Likely	Scalability, NUFFT
ExaSGD	Power grid	Tossup	Sparse Solvers
ExaStar	Astrophysics	Very Likely	OpenMP
ExaWind	Wind power	Very Likely	Sparse Solvers
GAMESS	Quantum chemistry	Very Likely	Portability
MFIX-Exa	Chemical reactors	Very Likely	Sparse Solvers
Subsurface	Subsurface flow	Likely	Sparse Solvers



	Chemistry/Materials						Energy Apps						Earth/Space Sci					Data				Nat Sec				
CD Center	LatticeQCD	NWChemEx	GAMESS	EXAALT	ExaAM	QMCPACK	ExaWind	Combustion-Pele	ExaSMR	MFIX-Exa	WDMApp	WarpX	ExaStar	ExaSky	EQSIM	Subsurface	E3SM-MMF	ExaSGD	CANDLE	ExaBiome	ExaFEL	ADTM LANL	ATDM LLNL	ATDM SNL	Tot	Pass
AMReX					🟨		🟩	🟩		🟩		🟩	🟩	🟩											6	3
CEED					🟩		🟩		🟩								🟦						🟩		4	3
CoPA				🟩	🟩						🟩			🟦											3	3
CODAR	🟦	🟦									🟩	🟨							🟨						1	3
ExaGraph		🟨					🟦											🟦		🟩				🟦	1	2
ExaLearn		🟦			🟦			🟦			🟦		🟦	🟩			🟦		🟨						1	2

🟩 Successfully integrated and on track for Frontier/Aurora

🟨 Integrated, may need work on functionality, performance, portability, and/or user adoption

🟦 Some exploratory proof-of-concept studies; exascale deployment may require additional time/$

Figure 1: Current KPP-3 metric for each Co-Design team

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AD	Applications Development
ALCF	Argonne Leadership Computing Facility
AM	additive manufacturing
Ames	Ames Laboratory
AMG	algebraic multi-grid
AMR	adaptive mesh refinement
ANL	Argonne National Laboratory
API	Application Programming Interface
ASC	Advanced Simulation and Computing
ATDM	Advanced Technology Development and Mitigation
CC	coupled cluster
CD	Co-Design
CFD	computational fluid dynamics
CFL	Courant-Friedrich-Lewy
CI	Continuous Integration
CLR	chemical looping reactor
CM	Chemistry and Materials Applications
COE	Center of Excellence
DAC	data analytics computing
DAO	data analytics and optimization
DEM	discrete element method
DFT	density functional theory
DMC	diffusion quantum Monte Carlo
DNS	direct numerical dimulation
DOE	US Department of Energy
DOF	degrees of freedom
EA	energy applications
EB	embedded boundary
ECP	Exascale Computing Project
EOS	equation of state
ESS	Earth and Space Science Applications
Fermilab	Fermi National Accelerator Laboratory

FFT	fast Fourier transform
FOM	figure of merit
FTE	faculty time and effort
HF	Hartree-Fock
HI	Hardware and Integration
HIP	Heterogeneous-Compute Interface for Portability
HPE	Hewlett Packard Enterprise
I/O	input/output
JLab	Thomas Jefferson National Accelerator Facility
JLSE	Joint Laboratory for System Evaluation
KPP	key performance parameter
L3	Level 3
L4	Level 4
LANL	Los Alamos National Laboratory
LBNL	Lawrence Berkeley National Laboratory
LCLS	Linac Coherent Light Source
LES	large eddy simulation
LLNL	Lawrence Livermore National Laboratory
M&S	modeling and simulation
MC	Monte Carlo
MD	molecular dynamics
ML	machine learning
NASA	National Aeronautics and Space Administration
NERSC	National Energy Research Scientific Computing Center
NESAP	NERSC Exascale Science Applications Program
NETL	National Energy Technology Laboratory
NIH	National Institutes of Health
NIST	National Institute of Standards and Technology
NNSA	National Nuclear Security Administration
NOAA	National Oceanic and Atmospheric Administration
NREL	National Renewable Energy Laboratory
NSCI	National Strategic Computing Initiative
NSF	National Science Foundation

OLCF	Oak Ridge Leadership Computing Facility
ORNL	Oak Ridge National Laboratory
PDF	probability distribution function
PIC	particle-in-cell
PNNL	Pacific Northwest National Laboratory
PPPL	Princeton Plasma Physics Laboratory
QCD	quantum chromodynamics
RANS	Reynolds-averaged Navier-Stokes
RANS/LES	Reynolds-averaged Navier-Stokes/large eddy simulation
SC	Office of Science
SIMD	single instruction, multiple data
SLAC	SLAC National Accelerator Laboratory
SME	subject matter expert
SMR	small modular reactor
SNL	Sandia National Laboratories
ST	Software Technology
UQ	Uncertainty Quantification
V&V	verification and validation
WBS	Work Breakdown Structure

1. OVERVIEW OF APPLICATION DEVELOPMENT

Exascale systems enable game-changing advances in scientific, engineering, and national security applications critical to the US Department of Energy's (DOE) mission. Such progress requires close coordination among exascale application, algorithm, and software development to address key challenges, such as extreme parallelism, reliability and resiliency, complex memory hierarchies, scaling to large systems, and data-intensive science. For selected critical problems, the ECP is creating and enhancing the predictive capability of relevant applications through the targeted development of requirements-based models, algorithms, and methods along with the development and integration of required software technologies in support of application workflows.

Given the broad US Department of Energy (DOE) and multi-agency demand for mission-critical modeling and simulation (M&S) and data analytics computing (DAC) applications, AD is contributing to the development of the ECP applications for DOE missions in science, energy, national security and the missions of other agencies, such as the National Institutes of Health (NIH), National Science Foundation (NSF), National Oceanic and Atmospheric Administration (NOAA), and National Aeronautics and Space Administration (NASA). For DOE alone, this scope encompasses full development support for application teams within the mission space of at least 10 DOE program offices. For other agency applications, the scope of AD is smaller and is one of partnership through the provision of selected staff to development teams in need of expertise in computer and computational science, applied mathematics, and high-performance computing (HPC).

To achieve these goals, AD includes six L3 Work Breakdown Structure (WBS) elements, each with multiple subprojects at Level 4 (L4). These are described in the following sections. Two KPPs, KPP-1 and KPP-2, are used to measure the success of the AD application projects; KPP-3 is used to measure the success of the AD co-design projects. KPP-1 performance is measured through a uniquely defined figure of merit (FOM) for each project. The meaning and requirements for each KPP are given in Table 2.

Table 2: KPP for ECP applications.

KPP ID	Description of scope	Threshold KPP	Objective KPP	Verification action/evidence
KPP-1	Performance improvement for mission-critical problems	50% of selected applications achieve FOM improvement ≥ 50	100% of selected applications achieve KPP-1 stretch goal	Independent assessment of measured results that threshold goal is met
KPP-2	Broaden the reach of exascale science and mission capability	50% of selected applications can execute their challenge problem	100% of selected applications can execute their challenge problem stretch goal	Independent assessment of mission application readiness
KPP-3	Productive and sustainable software ecosystem	50% of the weighted impact goals are met	100% of the weighted impact stretch goals are met	Independent assessment verifying threshold goal is met

1.1 Pre-exascale and Early Access Systems

In FY21, the AD projects had access to a selection of pre-exascale production, early-access, and pre-delivery hardware systems for testing as shown in Table 3. Each project has performed various levels of porting, optimization, and verification and validation (V&V) on a selection of these systems as described in the application project sections.

1.2 2021 AD Review

In the FY20 AD review described in *Map Applications to Target Exascale Architecture with Machine-Specific Performance Analysis, Including Challenges and Projections* [6] the projects were evaluated on three sets of charge questions for KPP-1, KPP-2, and Co-Design (CD) projects. Each set of questions was focused around performance on Summit and progress on the early access AMD and Intel platforms. For the FY21 review we supplied one set of charge questions for all projects as follows:

Table 3: Pre-exascale testing computing platforms.

System	Vendor	Center	Category	Notes
Summit	NVIDIA	OLCF	pre-exascale production	[1]
Tulip/Birch	AMD	Frontier COE	early-access	[2]
Spock	AMD	OLCF	pre-delivery	[3]
Crusher	AMD	OLCF	pre-delivery	[4]
Iris/Yarrow/Arcticus	Intel	JLSE	early-access	[5]

1. Walk through in detail your anticipated KPP threshold simulations (i.e. base challenge problem) along with the expected HPC resources required, as well as your plans for providing artifacts that allow an external reviewer to verify KPP completion. Specifically:
   (a) What simulations will you carry out on Frontier/Aurora to verify your KPP?
   (b) What artifacts will you provide from these simulations?
   (c) How do those artifacts confirm that the minimum requirements for the base challenge problem were met?
2. Provide a summary of your status on early hardware systems—Spock, Crusher and/or Arcticus—including performance results where possible.
3. Describe the main obstacles that stand between current status and successful completion of your KPP threshold on Frontier and/or Aurora. This discussion could include algorithmic challenges, system level issues, and the status of third party dependencies.
4. Explain any disparities between your original FY20–23 spending plan and your current spending status. For projects that are underspent, show revised spending plans through FY23Q3, including, when, who, and to what milestone(s) the carryover funds will likely be applied. In the event that the project is extended through FY24Q2 using contingency funds, comment on the top 2 or 3 priorities (e.g. code robustness, KPP stretch goals, etc.) you might address.
5. Briefly explain the expected status of your code(s) at the end of the ECP project. What is the minimum level of ongoing support (in terms of faculty time and effort (FTEs)) required to ensure productive use of the code(s) over the lifetime of the exascale machine(s)? What activities are most critical? Consider both a) code maintenance issues e.g. documentation, maintenance, support, porting, re-engineering, tutorials, outreach, and if appropriate b) extended algorithms and/or physics modeling to broaden the scope of problems that can be addressed.

An important component of this review was to introduce the external SMEs to the projects. As opposed to FY20, where the AD leadership team served as reviewers, in FY21 we recruited and assigned an SME to each project. These SMEs will serve as principal reviewers for their assigned projects for the remainder of the ECP. The role of the SME is to review and approve the final KPP measurements. To this end, each project has written specifications for their KPP benchmark computation that are reproduced in the first subsection for each project in this report. The KPP verification process is summarized as follows:

1. Each project defines their challenge problem benchmark specification and generated artifacts that confirm correct execution of the simulation;
2. Perform benchmark calculation on Frontier and/or Aurora;
3. Review of benchmark simulation by the project SME.

The details of this process are described in what is referred to as each subproject's *KPP Contract*, which must be evaluated in depth and approved by the lead SME. The process for approval is described in Fig. 2.

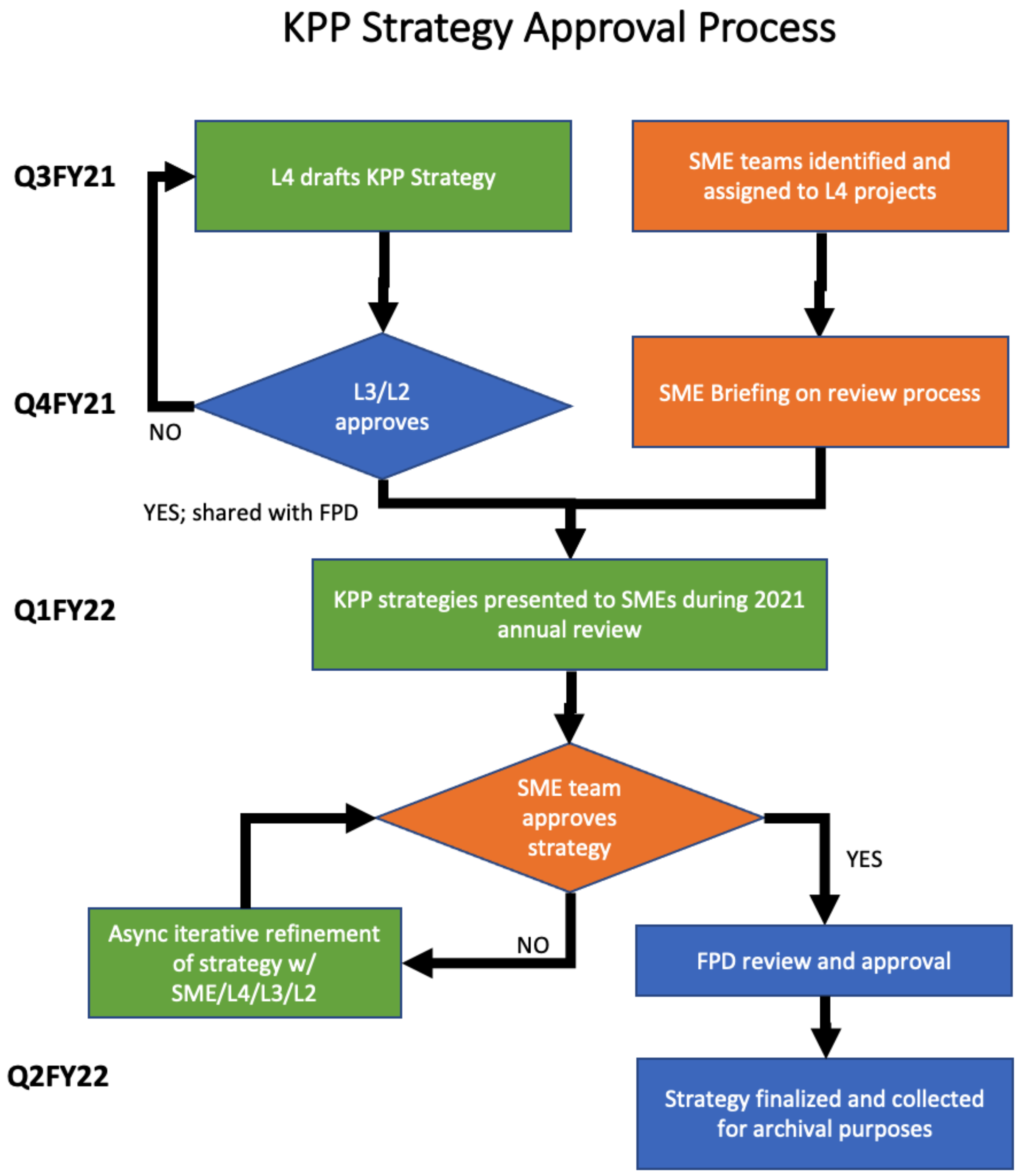

KPP Strategy Approval Process
Q3FY21
Q4FY21
Q1FY22
Q2FY22
L4 drafts KPP Strategy
SME teams identified and assigned to L4 projects
L3/L2 approves
NO
SME Briefing on review process
YES; shared with FPD
KPP strategies presented to SMEs during 2021 annual review
SME team approves strategy
YES
NO
Async iterative refinement of strategy w/ SME/L4/L3/L2
FPD review and approval
Strategy finalized and collected for archival purposes

Figure 2: Iterative process to finalize KPP contract.

Table 4: ECP applications targeting KPP-1.

Project name	Short description	Lead lab	Stakeholder program
LatticeQCD	Exascale lattice gauge theory opportunities and requirements for nuclear and high-energy physics	Fermilab	DOE NP, HEP
NWChemEx	Stress-resistant crops and efficient biomass catalysts	PNNL	DOE BER, BES
EXAALT	Molecular dynamics at the exascale	LANL	DOE BES, FES, NE
QMCPACK	Find, predict, and control material properties	ORNL	DOE BES
ExaSMR	Coupled Monte Carlo neutronics and fluid flow simulation of small modular reactors	ORNL	DOE NE
WDMApp	High-fidelity whole device modeling of magnetically confined plasmas	PPPL	DOE FES
WarpX	Plasma wakefield accelerator design	LBNL	DOE HEP
ExaSky	Cosmological probe of the Standard Model	ANL	DOE HEP
EQSIM	Seismic hazard risk assessment	LBNL	DOE NNSA, NE, EERE
E3SM-MMF	Regional assessments in earth systems models	SNL	DOE BER
CANDLE	Accelerate and translate cancer research	ANL	NIH

2. KEY PERFORMANCE PARAMETERS FOR AD

The AD focus area supports the development and evolved design of mission-critical science and engineering codes for efficient execution on exascale platforms. The ECP distinguishes between the code, which is typically a general capability, and an application, which uses the code to address a specific scientific or engineering question. A key concept is the definition of an exascale challenge problem. Each AD application code team must define an application challenge problem that is scientifically impactful and requires exascale-level resources to execute. Each exascale challenge problem targets a key DOE science or mission need and is the basis for quantitative measurements of success for each of the AD projects.

There are two measures of success used for AD application projects, referred to generically as the first and second of the ECP KPPs (KPP-1 and KPP-2). Projects are assigned exclusively to one of these two KPP groups, and each project is responsible for meeting the corresponding specific requirements. The KPP-1 applications (Table 4) and applications targeting KPP-2 (Table 5) are required to provide a detailed milestone plan that outlines all needed work to enable the successful execution of their exascale challenge problem. These milestone plans and the teams' progress in executing them are reviewed annually by AD leadership and external SMEs as part of the AD annual assessment. Progress toward KPP-2 is tracked between reviews with a dashboard to monitor timely milestone delivery. The KPP assignments were determined by the nature of the exascale challenge problem and the maturity of the individual code projects.

2.1 KPP-1

KPP-1 quantitatively measures the increased capability of applications on exascale platforms compared with their capability on the leadership-class machines available at the start of the project. Each application that targets KPP-1 is required to define a quantitative FOM that represents the rate of science work for their defined exascale challenge problem. FOM definitions are specific to an application area and are reviewed

Table 5: ECP applications targeting KPP-2.

Project name	Short description	Lead lab	Stakeholder program
GAMESS	Biofuel catalyst design	Ames	DOE BES
ExaAM	Additive manufacturing of qualifiable metal parts	ORNL	DOE NNSA, EERE
ExaWind	Predictive wind plant flow modeling	NREL	DOE EERE
Combustion-Pele	Combustion engine and gas turbine design	SNL	DOE BES, EERE
MFIX-Exa	Multiphase flow reactor design	NETL	DOE EERE
ExaStar	Demystify the origin of chemical elements	LBNL	DOE NP
Subsurface	Carbon capture, fossil fuel extraction, waste disposal	LBNL	DOE BES, EERE, NE, FE
ExaSGD	Reliable and efficient planning of the power grid	ORNL	DOE EDER, CESER, EERE
ExaBiome	Metagenomics	LBNL	DOE BER
ExaFEL	Light source-enabled analysis of molecular structure	SLAC	DOE BES
LANL ATDM	Ristra application	LANL	DOE NNSA
LLNL ATDM	MARBL multiphysics code	LLNL	DOE NNSA
SNL ATDM	SPARC for virtual flight testing and EMPIRE for electromagnetic plasma physics	SNL	DOE NNSA

internally and externally to the ECP to confirm that they are appropriate representations of capability improvements for that domain. Because exascale challenge problems cannot be executed on petascale resources, the FOM will typically account for differences in problem size, numerical precision, algorithm complexity, and physical model enhancement to allow for an accurate measurement of the ultimate FOM improvement used to satisfy KPP-1.

For KPP-1, one key concept is the performance baseline, which is a quantitative measure of an application FOM that uses the fastest computers available at the inception of the ECP against which the final FOM improvement is measured. This includes systems at the ALCF, NERSC, and OLCF, such as Mira, Theta, Cori, and Titan—systems in the 10–20 PFlops range. The expectation is that applications will run at full scale on at least one of these systems to establish the performance baseline. In cases where this is not possible, the largest scale run is scaled to the full system, assuming perfect parallel efficiency. A challenging situation arises when the final exascale challenge problem requires capabilities that did not exist at the start of the project (e.g., new code coupling, new physics models, or algorithmic approaches). For these applications, approximate estimates are constructed from individual code components with the expectation that the baseline can be refined, if necessary, once the new capabilities are in place.

Applications that target KPP-1 are required to demonstrate improvements to their FOM throughout the project on pre-exascale platforms. The teams' progress in improving their FOMs and preparing their codes for exascale architectures is reviewed annually by AD leadership and external SMEs as part of the AD annual assessment. Progress toward KPP-1 is tracked between reviews with a dashboard to monitor each application's current FOM increase against their performance baseline.

Because an exascale machine promises a rate approximately 50× the theoretical flop rate of the fastest currently deployed machine, the ECP sets the minimum FOM improvement aggressively at a factor of 50. The ECP supports complex multiphysics codes that put great demand on various aspects of the system: input/output (I/O) capacity and bandwidth, memory bandwidth, and memory latency (i.e., not just floating-point instruction capacity and throughput). Many applications are not based on algorithms that can perfectly use specialized hardware features. Aside from improved use of hardware use, one key focus of the ECP is the development of innovative algorithms that can achieve the same accuracy more efficiently. In cases for which new methodologies are developed (e.g., using lower complexity algorithms or reduced iteration counts), AD projects must demonstrate equivalent or better accuracy relative to the baseline approach. Incentivizing algorithmic advances is critical to the long-term impact of the ECP in the computational science community. Although risky, any projects that are successful in this approach could have FOM ratios much greater than 50. However, the final KPP-1 calculation gives no additional credit for measurements beyond the target value of 50, so one extreme success will not skew the overall project metrics.

FOM formulations were initially developed by the subproject leads and were iterated upon and vetted over a 2 year period. This includes a panel of external SMEs; technical ECP leadership across the entire management team; and experts at ALCF, NERSC, and the OLCF, as well as in ECP-wide meetings and previous project reviews. Furthermore, the KPP-1 definition, including threshold and objective targets, was modified from the original plans based on extensive external feedback.

The challenge problems defined by all KPP-1 and KPP-2 applications represent ambitious but realizable goals that take into account all the risks and uncertainty of such a complex project. Given the presence of accelerated schedules, highly specialized hardware, evolving software and application-level libraries, and open questions about programming models and compiler technology, some of the applications are likely to fall behind their initial schedule. On the other hand, if many anticipated risks are never triggered or are readily mitigated, some or even all the applications might achieve their individual KPP goals earlier than expected. This best-case scenario is accommodated by defining objective KPP values for each application subproject, which are based on stretch goals. *Stretch goals* are extended challenge problem definitions (i.e., challenge problems that require additional physics capabilities, code coupling, more complex geometries, or, in some cases, even larger problem sizes). Stretch goals represent a best-case scenario that requires many key pieces to fall into place, but they stand as critical definitions of the most ambitious realizable goals each project can envision within the scope of the current project.

Given the specialized nature of the hardware and the breadth and complexity of the application projects, it is highly unlikely that all KPP-1 applications will meet or exceed the target FOM increase. Therefore, the ECP sets the threshold value for project success at 50% of KPP-1 applications, which is determined by the ECP to be an ambitious but attainable goal and concurred with by the ECP's DOE sponsors. The objective value for KPP-1 is that 100% of the application subprojects meet or exceed their target FOM increase and also demonstrate their stretch goal. This objective value is considered an extremely ambitious goal that will further drive the science and engineering goals of ECP applications.

2.2 KPP-2

KPP-2 is intended to assess the successful creation of new exascale science and engineering DOE mission application capabilities. Applications that target KPP-2 are required to define an exascale challenge problem that represents a significant capability advance in its area of interest to DOE. These challenge problem targets are reviewed internally and externally to confirm that they are impactful, challenging, tractable, and of interest to a key DOE stakeholder. The distinguishing feature of KPP-2 applications relative to those that target KPP-1 is the amount of new capability that must be developed to execute the exascale challenge problem. Many KPP-2 applications lack sufficient code infrastructure from which to calculate an FOM performance baseline (e.g., they started in the ECP as prototypes). Without a well-defined starting point at the 10–20 PFlops scale, it is unclear what FOM improvement would correspond to a successful outcome. A more appropriate measure of success for these applications is whether the necessary capability to execute their exascale challenge problems is in place at the end of the project, not the relative performance improvement throughout the project.

Applications that target KPP-2 (Table 5) are required to provide a detailed milestone plan that outlines all the work needed to execute their exascale challenge problem successfully. These milestone plans and the teams' progress in executing them are reviewed annually by AD leadership and external SMEs as part of the AD annual assessment. Progress toward KPP-2 is tracked between reviews with a dashboard to monitor timely milestone delivery.

To quantitatively assess the successful completion of KPP-2, applications must demonstrate the capability to effectively use exascale hardware to execute their challenge problem. Because access to exascale resources might be limited and performance optimization might not yet be complete, KPP-2 applications can demonstrate exascale capability without running their challenge problem at full scale. This requires application teams to demonstrate: (1) parallel scalability sufficient to run at full exascale; (2) the ability to use specialized exascale hardware features, such as accelerators; and (3) the completion of all necessary physics and algorithmic capabilities to successfully perform the challenge problem. Internal and external review will confirm whether a team has satisfactorily met all three requirements. The metric for success is exascale capability; a code that runs on an exascale machine at the same rate or slower than on a pre-exascale machine will not be judged to be successful.

The ECP National Nuclear Security Administration (NNSA) applications are primarily focused on developing new and essential mission capabilities at exascale. All four NNSA ECP application projects target the ECP's KPP-2 metric. Because the national security nature of the NNSA challenge problems require a secure NNSA exascale computer (El Capitan), which will not be available before the ECP's current schedule to complete, progress and successful development of exascale capability by these applications cannot be assessed in the same way as the open DOE Office of Science (SC) applications. Instead, the ECP will leverage the NNSA Advanced Simulation and Computing (ASC) program milestone review and certification process by which these NNSA ECP applications will be assessed annually from FY19–23 for the necessary physics and algorithmic capabilities needed to execute their exascale challenge problems. The rigor of this process ensures that successfully completing these milestones through the end of the ECP does indeed verify that these applications can execute their exascale challenge problem once El Capitan enters its secure computing phase in FY24. The NNSA applications did not undergo a review in FY21, so no detailed results are reported in for these projects in this year's report.

The applications that target KPP-2 are working toward a significant advance in simulation capability (i.e., physics and numerical fidelity) in a relatively short time. As such, it is unlikely that all applications will be able to fully complete these ambitious objectives. Thus, the ECP sets the threshold value for project success at 50% of KPP-2 applications, which is determined by the ECP to be an ambitious but attainable goal and is concurred with by the ECP's DOE sponsors. Because the review and assessment criteria are slightly different for the NNSA applications, two out of the three must demonstrate exascale capabilities to meet the KPP-2 threshold.

Like KPP-1 applications, each KPP-2 application defines a stretch challenge problem that is above and beyond the baseline exascale challenge problem. The objective value for KPP-2 is that 100% of the application subprojects demonstrate their stretch challenge problem.

2.3 KPP-3 for Co-Design

KPP-3 is used to measure the impact of co-design software products and the projects in the ECP's Software Technology (ST) scope. ECP KPP-3 impact goals and metrics are the primary high-level means of connecting ECP co-design efforts to the ECP effort as a whole. Achieving these KPP-3 impact goals defines how the ECP's co-design centers are reviewed and how their success is determined.

A KPP-3 integration goal for co-design is defined to be its impact and use on its application customer codes, primarily the AD teams that are striving to meet KPP-1 and KPP-2 goals. The weights for the co-design center goals are determined by how many application teams rely on their software products. Of the co-design centers, AMReX and CEED are considered high-impact and will be assigned a weight of 2; CoPA is considered medium impact and will be assigned a weight of 1; and CODAR, ExaLearn, and ExaGraph are considered lower impact and will be assigned a weight of 1/2. This goal is explicitly tracked and reported for satisfying KPP-3 requirements.

Verifying the success of this goal will be documented as part of the capability and performance demonstrations on Aurora and Frontier needed to demonstrate completion of KPP-1 and KPP-2 objectives. In cases for which co-design capabilities are not explicitly required to meet KPP-1 and KPP-2 goals, separate integration runs will be performed for KPP-3 verification.

For all co-design centers, a passing score and a stretch goal on the number of applications that they will be integrated into and demonstrated on an exascale platform (Aurora and/or Frontier) have been defined. The KPP-3 threshold is defined as 50% of the products meeting or exceeding their weighted impact goals. The weighted impact stretch goal is the maximum reasonably achievable integration score for a co-design center if capability integrations are successful with all potential ECP applications on both ECP exascale platforms. The KPP-3 objective is that 100 % of the products meet or exceed their weighted impact stretch goals.

3. CHEMISTRY AND MATERIALS APPLICATIONS

End State: Deliver a broad array of science-based chemistry and materials applications that can provide, through effective exploitation of exascale HPC technology, breakthrough modeling and simulation capabilities that precisely describe the underlying properties of matter needed to optimize and control the design of new materials and energy technologies.

The Chemistry and Materials Applications (CM) L3 area (Table 6) focuses on simulation capabilities that aim to precisely describe the underlying properties of matter needed to optimize and control the design of new materials and energy technologies. The underlying physics that governs these application areas is computationally challenging. Capturing quantum effects can introduce significant communication nonlocality and computational complexity, for example. Because efficiently scaling these methods to exascale is likely to be difficult, a key assumption of the CM WBS L3 is that the L4 subproject leads already have significant experience with their methods and algorithms on petascale HPC resources and thus have a good understanding of where the biggest challenges to scalability are mostly likely to lie. The ECP is providing an essential catalyst to help propel these efforts forward so that key DOE priorities can be achieved. Given that this is a broad and very fundamental area of research with applications to many technology areas, it is understood that the ECP cannot provide exhaustive coverage of this area.

3.1 LatticeQCD

3.1.1 KPP Verification Plan

We are preparing the three major lattice quantum chromodynamics (Lattice QCD) code bases for exascale computing, namely, the MILC code [7], which emphasizes the highly-improved staggered-quark (HISQ) formulation, the CPS code [8], which emphasizes the domain-wall quark (DWF) formulation, and the Chroma code [9], which emphasizes the Wilson-Clover quark formulation, and two less widely used code bases, namely, the HotQCD code (unpublished) for high-temperature and density calculations, and QEX (unpublished) for

Table 6: Summary of supported CM L4 projects.

WBS number	Short name	Project short description	KPP-X
2.2.1.01	LatticeQCD	Exascale lattice gauge theory opportunities and requirements for nuclear and high energy physics	KPP-1
2.2.1.02	NWChemEx	Stress-resistant crops and efficient biomass catalysts	KPP-1
2.2.1.03	GAMESS	Biofuel catalyst design	KPP-2
2.2.1.04	EXAALT	Molecular dynamics at the exascale	KPP-1
2.2.1.05	ExaAM	Additive manufacturing of qualifiable metal parts	KPP-2
2.2.1.06	QMCPACK	Find, predict, and control material properties	KPP-1

Table 7: LatticeQCD HISQ challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	Meson decay constants and masses from first principles quantum chromodynamics.
Numerical approach and associated models	molecular dynamics, sparse matrix solution, deflation.
Simulation details	Generate part of a lattice ensemble with lattice spacing of 0.03 fm on a $192^3 \times 384$ grid with four flavors of highly improved staggered-fermion sea quarks at their physical masses but with degenerate up and down quarks. Specifically, run 2 molecular dynamics time units. Measure a standard set of meson decay constants and masses on those lattices.
Demonstration calculation requirements	A partially equilibrated lattice is needed, which is estimated to require at least a few dozen molecular dynamics time units of evolution. Measurements could be done on a single lattice.

exploratory studies of a wide variety of non-QCD field theories. Much of the portability across exascale architectures is provided by the lattice QCD QUDA [10] and Grid [11] software packages, which are being retooled for this purpose. We are also using OpenMP and LLVM.

Verification Simulations

Our FOM is based on six computations that represent three of the common fermion actions that USQCD currently uses: (1) the highly improved staggered quark (HISQ) action, (2) the domain wall fermion (DWF) action, and (3) the Wilson-Clover fermion action. The FOM for each action is determined from two standard benchmark components: the generation of a gauge configuration and a typical suite of measurements performed with that gauge configuration. Tables 7, 8, and 9 give details of the challenge problem.

HISQ

This benchmark performs the calculations needed to compute meson masses and decay constants. The two components measure (1) the rate of generating a new gauge configuration by using a molecular dynamics algorithm and (2) the rate of making a standard set of measurements on the gauge-field configuration. Specifically, we generate part of a lattice ensemble with lattice spacing 0.03 fm (femtometers) on a $192^3 \times 384$ grid with four flavors of HISQ sea quarks at their physical masses but with degenerate up and down quarks. For (1) we run for a time span of two molecular dynamics time units. For (2) we measure a standard set of correlators used to determine meson decay constants and masses on those lattices.

Table 8: LatticeQCD DWF challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	Study the decays of K, D, and B mesons. Examine simple single-hadron final states and more complex processes that involve multi-hadron final states, decay-induced mixing, long-distance effects, and electromagnetic processes.
Numerical approach and associated models	Use the methods of lattice QCD and a chiral fermion formulation. Electromagnetic effects are treated with infinite-volume methods. Linear and bilinear combinations of composite operators are renormalized non-perturbatively. Requires Lanzcos eigenvectors, deflation, all-to-all propagators, all-mode-averaging, open boundary conditions, and Fourier acceleration.
Simulation details	The target lattice volume is $128^3 \times 576$ with an inverse lattice spacing of $1/a = 3.7\,\mathrm{GeV}$ ($a = 0.053\,\mathrm{fm}$) using the Wilson gauge action and Möbius DWF.
Demonstration calculation requirements	MC evolution for 5 time units of the physical mass, $128^3 \times 576$, $1/a = 3.7\,\mathrm{GeV}$ ensemble. Start with a replicated configuration constructed from 512 periodic copies of an equilibrated $32^3 \times 72$ configuration. Perform a standard suite of measurements on a single configuration. These jobs should run on 1024 Aurora nodes in approximately 10 hours.

Table 9: LatticeQCD Wilson-Clover challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	Hadronic correlation functions and energies from QCD. On an ensemble of gauge fields, construct Euclidean correlation functions within many-body systems.
Numerical approach and associated models	Lattice QCD with a minimum grid size of $64^3 \times 128$ and lattice spacing of $0.091\,\mathrm{fm}$. Use the hybrid MC/molecular dynamics algorithm and sparse matrix solutions. Analysis methods will use multiple right-hand side solvers.
Simulation details	Generate part of an isotropic Clover ensemble with two light physical quark masses and one strange quark on a lattice size of $96^3 \times 256$ and a lattice spacing of $0.06\,\mathrm{fm}$. Specifically, run 10 trajectories (MC time units) to observe the behavior and stability of the algorithm.
Demonstration calculation requirements	A fully equilibrated lattice is needed. This will require about 1000 MC time units. Measurement tests can be done on a single lattice.

DWF

As with the HISQ action, two FOMs for the DWF component have been adopted. The first measures the rate at which an exascale-class gauge field ensemble can be generated, and the second calculates a suite of observable using this ensemble. The target lattice volume is $128^3 \times 576$ with an inverse lattice spacing of $1/a = 3.7\,\mathrm{GeV}$ ($a = 0.053\,\mathrm{fm}$) using the Wilson gauge action and Möbius DWF.

Wilson-Clover

The Wilson-Clover benchmark has two similar components. The first is the rate at which dynamical Wilson-Clover fermion lattices can be generated by using a molecular dynamics algorithm. Several solutions of the Dirac equation are computed and contracted to construct observables as part of the second component of the benchmark. In detail, we generate part of an isotropic Clover ensemble with two light physical quark masses and one strange quark on a lattice size of $96^3 \times 256$ and a lattice spacing of 0.06 fm. Specifically, we run 10 trajectories (MC time units) to observe the behavior and stability of the algorithm.

Method for Calculating the FOM

The FOM is the ratio of the exascale throughput (results per second) to the baseline throughput. The throughput is adjusted to correspond to the rate we would achieve if we ran multiple instances that filled the entire machine. The following values enter the calculation for each of the six components:

- $t_a(M)$ Time to complete problem a on the baseline machine M.
- $f_a(M)$ Fraction of machine M used for problem a.
- $t_a(E)$ Time to complete problem a on the exascale machine E.
- $f_a(E)$ Fraction of machine E used for problem a.

The basic FOM (without scaling up) for problem a is, then

$$\mathrm{FOM}(Ma \to Ea) = \frac{t_a(M) f_a(M)}{t_a(E) f_a(E)} \tag{1}$$

This formula measures the improvement in the rate of scientific output resulting from filling the entire machine with multiple instances of the calculation, each occupying the given fraction of the machine.

The exascale challenge problem is larger and more difficult, so we need to include a scaling factor W when comparing the timing of the larger problem b on the exascale machine with the smaller problem a on the baseline machine. (That fraction is optimized separately for each machine for the given problem.)

$$\mathrm{FOM}(Ma \to Eb) = \frac{t_a(M) f_a(M)}{t_b(E) f_b(E)} W(Ea \to Eb) \tag{2}$$

The scaling factor $W(a \to b)$ is based on timings on machine E and the number of flops, n_a and n_b required for each problem using the same algorithm.

$$W(Ea \to Eb) = \frac{t_a(E) f_a(E) n_b}{t_b(E) f_b(E) n_a}. \tag{3}$$

With this definition, if problem b is just two instances of problem a, this factor is exactly 1. (No performance improvement is involved in this trivial case.)

We use the same method for calculating the FOM (FOM_i) for each of the six component problems. The results are averaged to give the composite FOM:

$$\mathrm{FOM} = \frac{1}{6} \sum_{i=1}^{6} \mathrm{FOM}_i. \tag{4}$$

We run all six problems on both Aurora and Frontier. For the challenge problem, we use the larger FOM_i value from Aurora and Frontier for each component in the average. For the stretch problem, we calculate the composite FOM with all components taken from the same machine and use the smaller of the two composite FOMs.

Simulation Artifacts

HISQ

Gauge-field generation: We work with a $192^3 \times 384$ lattice. The input file contains parameter settings for running one trajectory of 2 molecular-dynamics time units

Analysis: We work with a $192^3 \times 384$ lattice. The input file contains parameter settings for carrying out a set of physics measurements.

The same set of measurements are used in the baseline and exascale calculation, except that the physical parameters are scaled appropriately. For both HISQ runs, the code generates a progress log of the run with timing information. The KPP is simply based on the logged time to completion.

DWF

The DWF demonstration calculation targets the $128^3 \times 576$ lattice volume with an inverse lattice spacing of $1/a = 3.7\,\text{GeV}$ ($a = 0.053\,\text{fm}$) using the Wilson gauge action and Möbius DWF. We will carry out a MC evolution for 5 time units beginning with a replicated configuration constructed from 512 periodic copies of an equilibrated $32^3 \times 72$ configuration. In addition, a standard suite of measurements will be run on a single configuration. These jobs should run on 1024 Aurora nodes in approximately 10 hours.

In addition to input run parameters we will generate this demonstration ensemble starting from a $32^3 \times 72$ equilibrated ensemble which requires the input of a 1.5 GB file. The final values for the two DWF FOMs can be computed directly from the logged execution timings. We would also output the resulting gauge configuration for future use, a 1.5 TB file.

Wilson-Clover

Gauge-field generation: We work with a $96^3 \times 256$ lattice. The Clover gauge generation demonstration uses a $N_f = 2 + 1$ clover dynamical fermion action and Symanzik improved gauge with lattice space $1/a = 0.06\,\text{fm}$. The XML input file with parameter settings for running 10 trajectories will be provided. An equilibrated lattice should be provided on input to give a reasonable estimate of total performance. If such an equilibrated configuration is not available, a replicated copy of a hot start will be used. It is estimated that about 50 Frontier node hours will be required for the test runs on a 1024 node partition.

Analysis: We work with a $93^3 \times 256$ lattice. The analysis step involves computing 12 right-hand sides sets of Clover-Dirac solution vectors and constructing a propagator object in memory. This object will be contracted to produce Euclidean correlation functions which are very small in size. The input is a lattice gauge configuration from the gauge generation step.

The same set of measurements are used in the baseline and exascale calculation. The gauge generation step will generate a lattice gauge configuration output file of 125 GB in size. The standard output from the task holds the timings for all of the steps. The analysis step uses one of these configurations. The timings for the inversions and the contractions are part of the standard output. The Clover FOM can be calculated from these timings.

3.1.2 Early Hardware Status

Since we rely heavily on the QUDA and Grid packages to achieve portability and efficiency, we include them in this status report in addition to the three major code bases that contribute to the FOM. At the time of this writing, Crusher was not available to the ECP community, so we discuss only status on Spock and Arcticus.

QUDA

QUDA runs natively on NVIDIA GPUs. The port to Heterogeneous-Compute Interface for Portability (HIP)/AMD is well in hand and gives decent performance on Frontier testbeds. The SyCL and OpenMP ports are progressing.

Grid

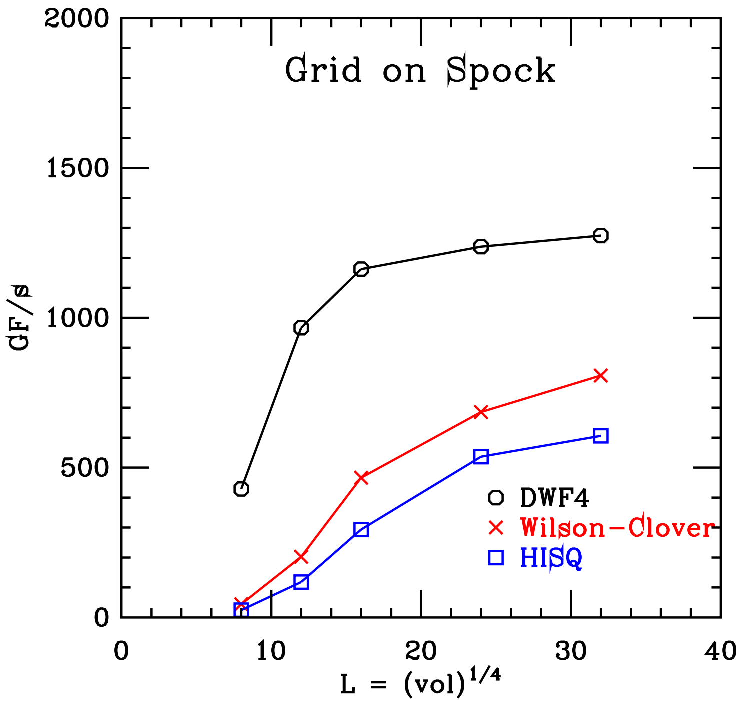


Figure 3: Performance of the single-precision Grid Dslash code on a single Spock GPU for the three fermion actions.

The port to SyCL is giving good results on Intel discrete GPUs. The HIP port is being optimized. The Grid code is being tested on Arcticus and Spock. Performance of Grid on a single Spock GPU as function of problem size is shown in Fig. 3 for all three fermion Dslash algorithms. The DWF action used $L_s = 12$.

HISQ

The HISQ calculation uses the MILC code [7]. The computationally demanding parts of the calculation are carried out by parts of the QUDA and Grid packages, which support the NVIDIA, AMD, and Intel GPUs. With QUDA/HIP MILC runs reasonably well on Frontier testbeds, but needs more attention to optimization. We are still working on the Grid/SyCL route to Aurora and are waiting eagerly for the QUDA/SyCL port for the components needed for lattice generation. A small version of both components of the HISQ FOM have been run on Spock. The code has also been built on Arcticus where it accesses the Intel GPUs via both QUDA and Grid.

DWF

The DWF calculation uses CPS code. Like the MILC code, it has interfaces to Grid and QUDA and expect to use them on both Frontier and Aurora. OpenMP is used for routines outside QUDA or Grid.

Wilson-Clover

The Wilson-Clover calculations use the Chroma and QDP++ code. They access GPUs through QUDA and Chroma-JIT. The latter is custom code developed at Jefferson Lab that does just-in-time compilation and uses LLVM to achieve portability. (LLVM is part of the compiler toolchain for NVIDIA, AMD, and Intel.) This code was recently complete. The Chroma code gives decent performance on Spock. Work on Aurora testbeds is underway.

The Thomas Jefferson National Accelerator Facility (JLab) Chroma Wilson-Clover code is farthest along in testing on AMD GPUs. In the performance examples shown here, the code address the AMD GPUs through the HIP port of the QUDA library. Since our other codes can also access the AMD GPUs via QUDA, the performance of QUDA for the Wilson-Clover code is a good indication of the expected performance of our

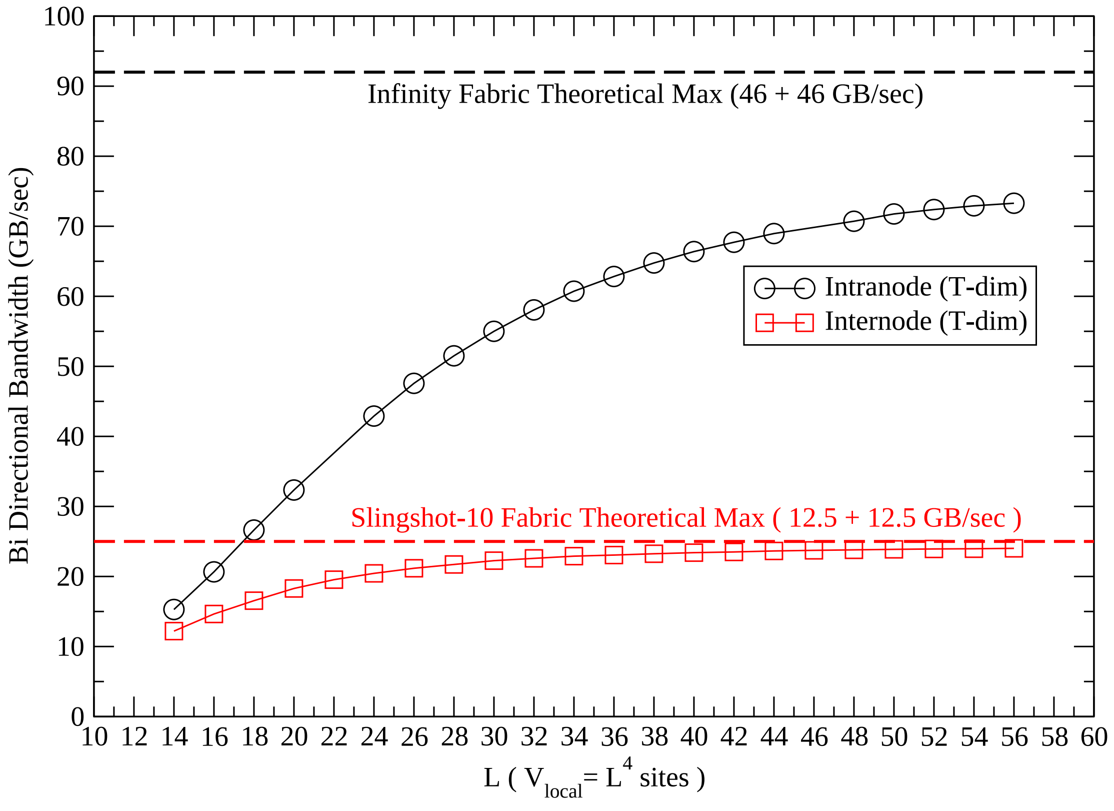


Figure 4: Bidirectional bandwidth for the QUDA "Halo" exchange on Spock *vs.* problem size. Black symbols and line: Intranode bandwidth. Red symbols: internode between two nodes. The dashed horizontal lines indicate the theoretical maximum bandwidth.

other codes. Figure 4 shows the bidirectional bandwidth on Spock for "Halo exchange" within and between Spock nodes as a function of problem size. With larger problems, message-passing latency is amortized and QUDA nearly saturates the bandwidth. Since the performance of our codes tends always to be limited by the internode network, this result is a good indicator of overall performance.

3.2 NWChemEx

3.2.1 KPP Verification Plan

The project base challenge problem objective focuses on the optimization of feedstocks for the efficient production of biomass for biofuels and other bioproducts on marginal lands. The first focal point for the NWChemEx project is transport across cellular membranes in response to biotic and abiotic stresses. Membrane transporters form gates between cells and the environment for the flow of metal ions as well as carbon, nitrogen, nutrients, and metabolic products and are key modulators of stress. An example is the Bax inhibitor that controls the transport of Ca2+ in transgenic sugarcanes. The process driving trans-membrane transport in the Bax inhibitor is poorly understood, although from experimental studies the mechanism appears to be proton controlled and involves two active sites, with one of those sites undergoing large conformational changes on protonation. It is critical to have a detailed molecular understanding of transport processes involved in stress responses to develop genetic modifications that lead to better stress-resistant crops.

Describing proton-controlled ion Ca2+ transfer in the Bax inhibitor in its local cellular environment requires modeling of hundreds of thousands of atoms to describe a suitable portion of the cellular membrane, the 3500 atom Bax inhibitor-1 protein, as well as a sufficient region of the immediate cytoplasmic environment.

Table 10: NWChemEx challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	Solution of the electronic Schrödinger equation to predict the structures and energetics of the reactants, products, and transition states involved in the conversion of transport pathways.
Numerical approach, and associated models	Hartree-Fock, density functional theory, and coupled cluster theory (both canonical and reduced scaling versions). Coupled Cluster theory is required to achieve an accuracy of 1 kcal/mol or better in the prediction of the energetics of molecular interactions, including barriers to chemical reactions.
Simulation details	Run calculations on fragments of the ubiquitin molecule, which is typical of proteins like the Bax inhibitor. The fragments begin with DGLRT, the 79-atom system used to benchmark NWChem, and end in ubiquitin (a 1,231 atom system to be run on Frontier and Aurora).
Demonstration calculation requirements	Iterative solution of the coupled cluster single and doubles (CCSD) equations followed by the calculation of the perturbative triples (T) correction. The latter is the most numerically intensive, time-consuming part of the CCSD(T) calculation and has a well-defined dependence on the computational details (number of electrons, number of occupied orbitals, number of virtual orbitals, etc.). Both the CCSD and (T) correction will be used to define the FOM.

Currently proton-controlled transport simulations can only be performed using standard force fields that lack a description of the proton transfer process. Truly predictive modeling of this molecular system requires use of high-level quantum mechanical methods, describing 10^3–10^4 atoms with coupled cluster (CC) methods embedded in an environment of 10^5 atoms described with density functional theory to parameterize the proton hopping processes with chemical accuracy for subsequent use in, for example, simulation of proton dynamics using adaptive force fields and molecular dynamics and long-time conformational sampling at timescales of milliseconds of protonated and de-protonated states.

The NWChemEx challenge problem details are listed in Table 10.

Verification Simulations

The final demonstration calculation will be the calculation of the reduced scaling CCSD(T) on Frontier (initially) and Aurora effectively using 100% of the machine. The molecule will be at least the size of the ubiquitin protein. The results of the demonstration calculation, the early Titan results, and the scaling based on molecular sizes and machine scaling will be used to determine the final KPP result.

Facility Requirements

We are not anticipating any specific or significant libraries that we do not have access to through the current stack or freely available software. We will need to have access to 1000–1500 nodes for performing scaling runs that will feed into the extrapolations from the Titan results.

Input Description and Execution

The chemical systems that will be used for the demonstration calculations are stored in a GitHub repo at https://github.com/NWChemEx-Project/NWX_TA/tree/master/HUb_1UBQ. The main inputs for the simulations require three pieces of information:

- the molecular structure
- the basis set
- the method/energy expression to evaluate

With regard to these quantities the setup is a bit different for NWChem and NWChemEx. NWChem inputs will include the molecular structure through a “load” directive in the geometry input block. The basis set and the method are given in the input file. In NWChemEx the molecular structure and basis sets of the relevant test cases will be available in the application. Based on this, all three inputs can be selected with keywords in the input file. The planned simulation is a minimum of the size of ubiquitin (1231 atoms) run with NWChemEx using DLPNO CCSD(T) using the cc-pVTZ basis set as a minimum (aug-cc-pVTZ will be used if the calculation will converge). Canonical CCSD(T) calculations have been performed using NWChem with a small fragment of ubiquitin on Titan as the baseline (79 atoms, 1243 basis functions). The scaling of NWChem to larger systems and the full machine have been evaluated by running NWChem on Summit for the CCSD and the (T) independently since the scaling of the two parts of the calculation are quite different. This scaling will be used to extrapolate a simulation of a full run of NWChem on the full exascale machine for the final molecular system (ubiquitin at a minimum). NWChemEx canonical CCSD(T) calculations will be run on a ubiquitin fragment of 100–200 atoms to show the improvement from the use of GPUs in NWChemEx over the implementation in NWChem for the canonical method. These latter simulations will only be for information purposes as the final FOM will be the FOM for the DLPNO CCSD(T) code for NWChemEx divided by the FOM for the extrapolated canonical CCSD(T) for NWChem.

The number of virtual orbitals for the canonical CCSD(T) is the number of basis functions minus the number of electrons divided by two (two electrons in each occupied molecular orbital). Thus, the canonical CCSD(T) uses the full virtual space. In the DLPNO version we are using a well-localized, occupied pair specific virtual space, which is much much smaller than the original virtual space. We go from the full virtual space (hundreds of thousands) to domain-specific projected atomic orbital (PAO) space (hundreds to thousands) and finally to the pair natural orbital (PNO) space (tens to hundreds).

In contrast to published works we focus on less crude approximations—for large systems we need more than 99.9% of recovered correlation energy, so tighter thresholds leads to higher accuracy, higher cost, but still much cheaper than N^6, i.e. around N^2–N^3. We plan to work with larger PNO spaces and drop a fewer number of localized molecular orbital pairs. This is mainly because our code will work on supercomputers, where we can afford more resources (memory, etc) in order to achieve more accurate results than an $O(N)$ DLPNO-CC implementation.

While we can run the entire calculation in memory, in practice, for restart purposes, we would need to store various tensors on disk during or post CCSD. In order to estimate the data storage requirements we consider the most demanding intermediates created during the DLPNO-CCSD/CCSD(T) calculations, which are 3-index integrals in the PAO space, precomputed integrals in PNO-specific spaces, transformation matrices between PAO and PNO space, CCSD amplitudes and residuals in the PNO space. For calculation of the (T) correction, we also consider the estimated size of 6-index intermediates in the triples-natural orbitals (TNO) spaces and transformation matrices between PNO and TNO space. While the most expensive integrals and transformation matrices for DLPNO-CCSD algorithm will be computed and stored at one time in the pre-DLPNO part of the code (before amplitude equations execution), for the (T) correction the data will be calculated after DLPNO-CCSD convergence. Based on the ubiquitin fragment tests, for the full ubiquitin molecule for DLPNO-CCSD(T), we estimate the upper bound for storage requirements to be 880 TB.

Simulation Artifacts

The codes produce regular text output files. The output contains key pieces of data showing the progress of the calculation, such as main stages of the computation, the energy at each iteration in iterative methods, as well as timing information needed to understand the performance of the code. At present the performance data is extracted by hand and inserted in a spreadsheet to calculate the KPP. However, it is possible to write some scripts to extract the relevant data and compute the KPP.

In addition to the human readable output files there are some quantities that need to be passed from the CCSD to the CCSD(T) calculation. The size of the data passed is implementation dependent and bigger in NWChem than in NWChemEx. Overall, the data is of the order of $O(N^4)$ where N is proportional to the number of basis functions. This data is stored in binary scratch files.

The FOM and KPP will be calculated as detailed in the reports associated with JIRA ADSE11-148 and its sub-reports. Ideally the performance comparison would be based on comparing the NWChem and NWChemEx results for the same chemical systems (at least the size of ubiquitin with an aug-cc-pVDZ basis set, $\sim$ 39,000 basis functions). However, it is clear that this will not be possible with NWChem. The method

Table 11: NWChemEx CCSD and (T) performance results on Spock and Summit. All timings in s.

	Nodes, GPUs/node	(T)	CCSD per iteration	CCSD total
Spock MI100	1,4	286	18.3	404
	2,4	150	10.5	239
	3,4	104	8	186
	4,4	80	7.6	178
Summit V100	1,6	538	27	598
	2,6	270	10.4	236
	3,6	188	7.9	181
	4,6	147	6.8	160

for extrapolation to be used is described above.

Confirmation of correct results will be accomplished through sanity checks based on comparison with canonical codes for small and medium-sized systems for which the canonical calculations can be performed. We plan to perform the following checks:

- For a given chemical system, check if the DLPNO CCSD(T) energy exactly matches the canonical CCSD(T) energy when all DLPNO thresholds are set to zero (full spaces).
- Estimate the amount of correlation energy that can be recovered (ΔE_{corr}) with respect to the DLPNO thresholds. We will calculate profiles for each threshold separately and then for combined cases in order to determine optimal settings (maximum correlation energy recovered vs computational cost).
- For large molecules we will estimate optimal DLPNO thresholds (for pairs, domains, PAO space, PNO spaces) by information based on the analysis described above.

3.2.2 *Early Hardware Status*

We have made significant strides toward enabling our software on multiple platforms, with Spock and Joint Laboratory for System Evaluation (JLSE) being the prime platforms targeted for development. We have made significant strides on all of the major time consuming computations toward HIP and SYCL ports. All of the code compiles on JLSE and Spock, although there are still parts of the code that have not been fully ported to the GPUs.

TAMM [CCSD(T)]

The canonical CCSD and CCSD(T) calculations were performed with the DGRTL fragment with the STO-3G basis (231 basis functions) on the AMD and Intel GPU testbeds, Spock and JLSE. The DLPNO CCSD and DLPNO CCSD(T) codes were built with the appropriate latest SDKs and checked for numerical correctness using a few small test cases on a single node. One focus of the work next year is to optimize the DLPNO algorithms on these architectures.

The canonical CCSD and (T) calculations were performed on up to 4 nodes of Spock where each node has 4 AMD MI100 GPUs. Table 11 shows comparison with equivalent node configurations on Summit where each node has 6 NVIDIA V100 GPUs. We observe a performance improvement of approximately 1.8× for the (T) calculation compared to Summit. The CCSD calculation takes 21 iterations to converge and the performance is similar to Summit except when run on a single node. This is due to the block sizes used on GPUs for contractions. A single CCSD contraction is split into individual tasks where each task computes a block of the output tensor. Larger block sizes improve GPU performance but significantly increase the communication costs when run at a larger scale. We plan to explore batching of these blocks for improved GPU performance for CCSD.

TAL-SH

TAL-SH is the on-node tensor algebra library that supports multi-core CPU processors. and GPU accelerators, including multi-GPU support. All previous AMD ROCm runtime issues have been resolved. TAL-SH is being optimized on Spock and being used in the TAMM benchmarks listed above. Tensor-hypercontraction (Hadamard support) is currently in progress as it is useful for the DLPNO implementation. SYCL support is planned using LibreTT (discussed below) and additional porting efforts.

TiledArray

The team has converted most of the TiledArray code from CUDA to SYCL, which includes two TiledArray dependencies, Umpire and LibreTT, and TiledArray CUDA kernels. The memory management library, Umpire, and the tensor transpose library, LibreTT, incorporate CUDA and SYCL support, compile using the latest Intel oneAPI SDK, and pass their accompanying unit tests when run on the latest Intel testbed. The conversion of TiledArray CUDA kernels involved substitution of cuBLAS library calls with oneMKL analogs. In addition, Intel oneDPL library calls replaced CUDA Thrust functions in the TiledArray code. The resulting TiledArray code integrates multiplatform versions of the Umpire and LibreTT libraries as well as CUDA and SYCL versions of the TiledArray GPU kernels. The enhanced TiledArray CMake compilation script handles compilation for both CUDA and SYCL platforms. The TiledArray library compiles for the latest Intel testbed by using the oneAPI SDK. Most of the TiledArray unit tests and examples compile as well. Fock matrix assembly, a non-GPU test, runs to completion, demonstrating basic integrity of the revised TiledArray code. A TiledArray tensor transpose test that invokes the LibreTT library call runs on Intel GPU. Effort is under way to incorporate TiledArray Continuous Integration in GitLab on Argonne JLSE. HIP support is underway.

DFT

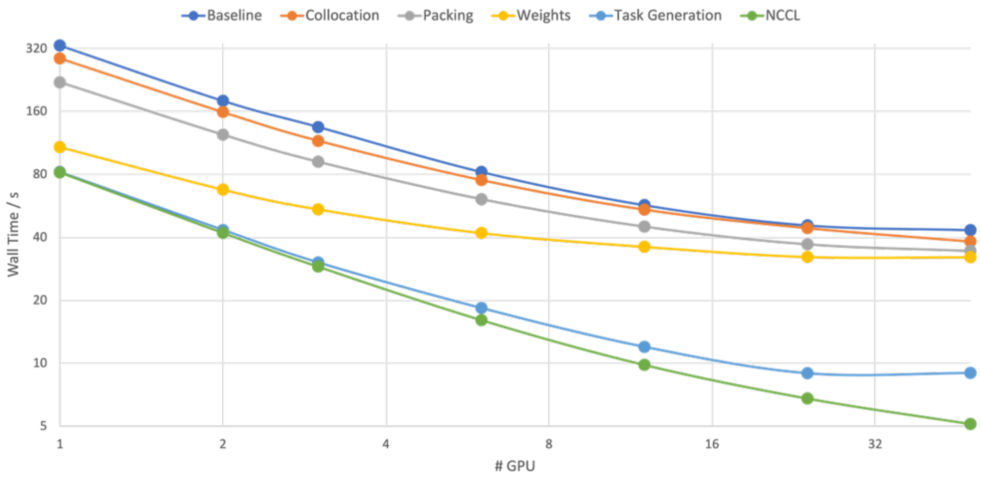
As of this review, we have CUDA, HIP and SYCL ports of the density functional theory (DFT)-XC integrator. There has recently been a large structural refactor to allow for porting to new architectures (per the algorithmic design outlined in [12]) which allows the decoupling of high-level algorithmic workflows from the implementation details of performance critical kernels to be optimized for each architecture of interest. However, due to lack of resources, only the CUDA and HIP ports of the integrator have been ported in this new software infrastructure as of this report, the SYCL functionality will be ported over in the coming fiscal year.

The CUDA port of the XC integrator is running on both Oak Ridge Leadership Computing Facility (OLCF) (Summit) and National Energy Research Scientific Computing Center (NERSC) (Cori-GPU and Perlmutter) resources with all unit tests and performance tests giving correct results. As a part of the NERSC-NERSC Exascale Science Applications Program (NESAP) program, we have had substantial collaboration with NVIDIA development technicians to optimize the CUDA implementation on contemporary platforms (V100/A100). This collaboration has produced the following optimizations:

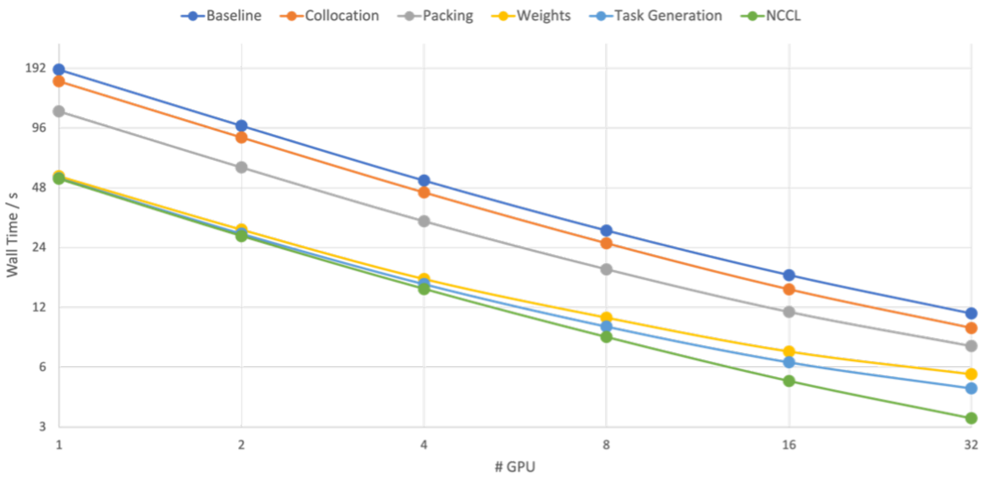
- Optimization of the collocation kernel which evaluates the Gaussian basis functions on the quadrature grid;
- Optimization of the packing kernel which brings non-contiguous data into contiguous buffers in device memory for use with batched level-3 BLAS operations;
- Optimization of the partition weights kernel which enables the construction of molecular quadratures as a union of atomic quadratures;
- The development of a device kernel for the generation of the integration tasks to enable a static load balance in distributed memory. In the initial implementation, this was originally only performed on the host and its constant cost severely limited strong scalability, especially on Summit;
- Optimization of distributed memory reduction via NCCL/NVSHMEM - this drastically improved strong scalability by reducing the communication overhead.

With these improvements, we have achieved an 8× improvement within this fiscal year, yielding $O(40–80\times)$ over the NWChem reference timings.

The HIP port is currently running on Tulip and Spock with all unit tests passing. The small (Benzene PBE0/cc-pVDZ), medium (Taxol PBE0/6-31G(d)), and largest (Ubiquitin PBE0/6-31G(d)) performance



(a) Summit (6 × NVIDIA V100).



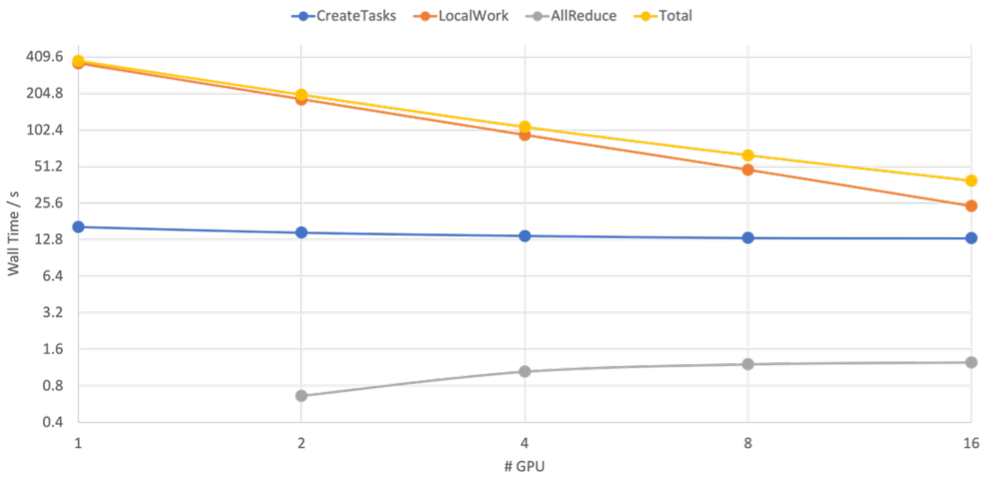
(b) Perlmutter (4 × NVIDIA A100).

Figure 5: Kernel optimizations for Ubiquitin RPBEO/6-31G(d), $N = 10{,}292$ on Summit and Perlmutter.

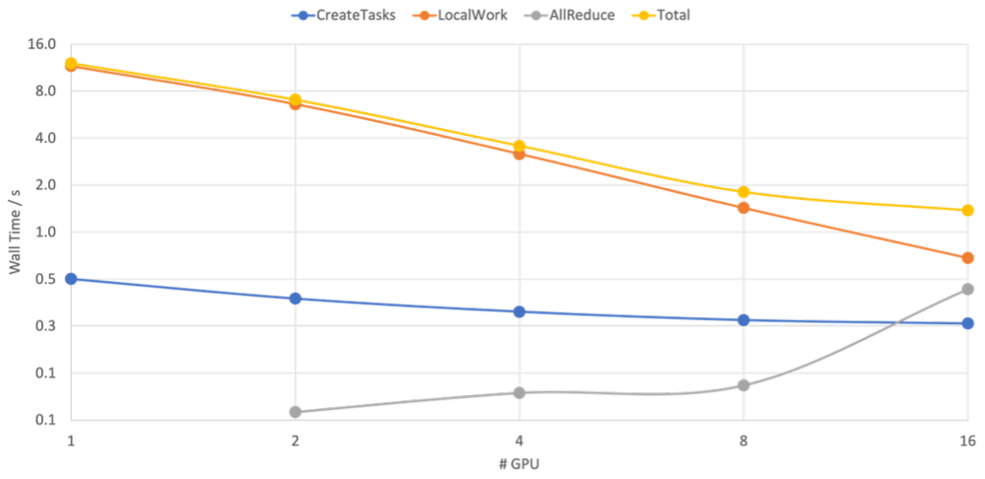
tests can be run successfully. As the kernel-level optimizations developed for the NVIDIA platforms (see Fig. 5) rely heavily on vendor-specific programming utilities (inline PTX, warp intrinsics, etc), their direct translation into HIP is not possible. In addition, the ROCm analogy of NCCL (RCCL) had not yet been integrated. Analogous optimizations for AMD platforms are currently under development and will be prioritized under the "optimization for exascale platforms" milestone once we have access to Crusher and AMD MI200 GPUs which will comprise Frontier. Interestingly, the current results (see Fig. 6) are $O(8\times)$ slower than the A100 on the MI100, which is precisely the difference between the A100 before and after optimization. That bodes well for putting time into these optimizations.

DLPNO

In FY 21, our main focus was the initial implementation of DLPNO CCSD and DLPNO (T) methods. In the context of these reduced scaling methods, the tensors to be contracted exhibit significant sparsity arising from the exploitation of local interactions in the orbital space. Thus, instead of a uniform set of large block-block contractions needed with traditional implementations using dense symmetric tensors, a much larger number of small tensor contractions is required, where the sizes of the dense tensor blocks are variable. For supporting efficient tensor operations in the context of reduced scaling methods, we have implemented a



(a)



(b)

Figure 6: Kernel optimizations for (a) Ubiquitin RPBEO/6-31G(d), $N = 10{,}292$ and (b) Taxol RPBEO/6-31G(d), $N = 1013$ on Spock ($4 \times$ AMD MI100).

pre-screening which extracts the non-zero blocks within the sparse tensors used in these methods.

This initial implementation allowed us to represent the sparse problem as contractions on dense blocks of a tensor which can already be executed efficiently on GPUs. This formulation avoids the need for special care of the contraction kernels as well as the communication of small block tensors that can quickly become a bottleneck over the full computation. With the new formulation, we focused on implementing a code generator prototype that employs an operation minimization algorithm for a single expression to find the optimal contraction ordering. With this tensor operation generator, users do not have to manually specify the best performing binarization for any expression. Our code generator automatically finds the best performing binarization for the tensor contraction chain and produces the corresponding tensor operations. We also intend to extend this prototype to explore possible optimizations over the full computation by defining intermediates that are commonly used in multiple tensor contraction expressions. This enables exploration of additional optimizations by extracting these (read-only) intermediates outside the computation loop allowing re-use and eliminating any redundant computation.

We developed a code generator prototype that focuses on the higher-level optimizations specifically for reduced scaling (DLPNO) methods. This prototype allows generating tensor operations that can be executed on different vendor accelerators through TAMM's unified execution interface. For the initial implementation, we focused on extracting sparse tensor blocks for the DLPNO methods to construct dense index spaces describing the data involved in the computation. With this formulation, we were able to make use of existing execution schemes in TAMM that use state-of-the-art libraries tuned to run on different vendor accelerators. We currently have limited support for tensor contractions with Hadamard indices in TAL-SH which prevents execution of such contractions on AMD and NVIDIA GPUs. A significant portion of the tensor contractions used in the new DLPNO formulation involve Hadamard indices which make this support crucial for further optimizations. We are working with the TAL-SH developers to address this issue.

In the final quarter of FY21, we started focusing on the optimizations over the initial implementation of DLPNO-based methods. We are following a multi-layered optimization scheme where we apply various optimizations on different levels of abstractions. As the first optimization effort, we started implementation of optimization passes to our code generator that generates the equations with different space configurations (i.e., integrals in PAO form, PNO form, etc.). This allows us to explore different optimization options on the code generator depending on the different problem sizes. As another optimization on a lower-level abstraction, we started exploring sparse data representations to efficiently store and operate on sparse tensors. Finally on the lowest level, we will be exploring kernel-level optimizations that can make use of efficient sparse tensor operation implementations.

3.3 GAMESS

3.3.1 KPP Verification Plan

The team will compute the chemical energetics on a model reaction with representative mesoporous silica nanoparticles (MSN). An adequate representation of the MSN pore requires thousands of atoms, including solvent, with an appropriate basis set. This could require tens to hundreds of thousands of basis functions.

The energy surface will be mapped via GAMESS calculations by using the effective fragment molecular orbital (EFMO) + RI-MP2 methodology with refined calculation by using the EFMO plus the RI-coupled cluster approach or possibly GAMESS EFMO + QMCPACK (§ 3.6) quantum Monte Carlo (QMC)approach (as stretch goals) in the reaction region for more accurate energetics. Here, RI is resolution of the identity, and MP2 refers to second order perturbation theory.

The GAMESS challenge problem details are listed in Table 12.

Verification Simulations

Facility Requirements

The GAMESS code base requires both C++ and Fortran OpenMP offload for the GAMESS Fortran code path. For exascale resource requirements, the team will use Summit benchmarks plus MI100 performance results to estimate Frontier requirements (and similar on Aurora).

Input Description and Execution

Table 12: GAMESS challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	MSN fragment energetics (reactants vs. products) and dynamics (diffusion rates) computations with at least 10,000 atoms for the pore + solvent. The go-to level of theory will be EFMO/RI-MP2 with an adequate basis set. The solvent will be treated with the same level of EFMO/RI-MP2 theory or with EFP. Final energies (stretch goal) will be captured using multilevel EFMO calculations with coupled cluster calculations for the reaction region and RI-MP2 elsewhere.
Numerical approach and associated models	Configurations can be computed concurrently; Each configuration will use the EFMO fragmentation approach to parallelize the calculation of underlying quantum-chemistry methods, which are typically characterized by dense linear algebra-like operations: Hartree-Fock, RI-MP2, RI-coupled cluster/QMC.
Simulation details	At least 10,000 atoms, comprising the MSN pore, reactants, and solvent molecules, an estimated million basis functions. Approximately four times this size for the stretch goal.
Demonstration calculation	Demonstrate the ability to complete the requirements science challenge problem by running a subset of atomic configurations concurrently with EFMO-RI-MP2 on the full exascale system.

We need coordinates of atoms, types of atoms (atomic numbers), basis set specification, level of theory (e.g., HF, RI-MP2, and so on), specification of initial guess of orbitals, convergence criterion (default or other), type of converger (default or other), and required number of nodes/cores. The plan is to upload input files that can be reviewed by the committee to meet the minimum criteria described in the Table 12, "Simulation Details". Inputs for base problem have been uploaded to github repository at https://github.com/gms-bbg/fragmentation-inputs.

Simulation Artifacts

Output files contain total energy, fragment identification, fragment energies, orbitals, and orbital energies. Structures and orbitals can be visualized by the companion code MacMolPlt or other visualization software such as Avagadro.

Verification of KPP-2 Threshold

The team will verify the libcchem C++ code and offloaded GAMESS Fortran code by comparing results of modest sized calculations with the original GAMESS application, which is known to be working and untouched by ECP. GAMESS itself has been vetted and verified for years. Accuracy of the fragmentation calculations has been vetted in GAMESS with accuracy superior to 2 kcal mol^{-1}. Again, implementations in libcchem and offloaded GAMESS Fortran will be verified against previously verified implementations, such as the currently distributed GAMESS code.

The successful production of GAMESS output as delineated under problem artifacts can be provided as CPU versus GPU performance results for modest size runs of a small fragmented system. It would help if facilities can provide percent of GPU used for the run.

3.3.2 Early Hardware Status

Before proceeding to discuss our experiences on Spock, Crusher and Arcticus, it is informative to provide an overview of our general approach and to provide evidence that, since we demonstrate that we can efficiently and effectively make use of at least 90% of Summit (see the Figs. 7 and 8), we are confident that the same will be true for the new systems once they are actually ready for prime time. As the reader will appreciate from our responses below, prime time has not yet arrived.

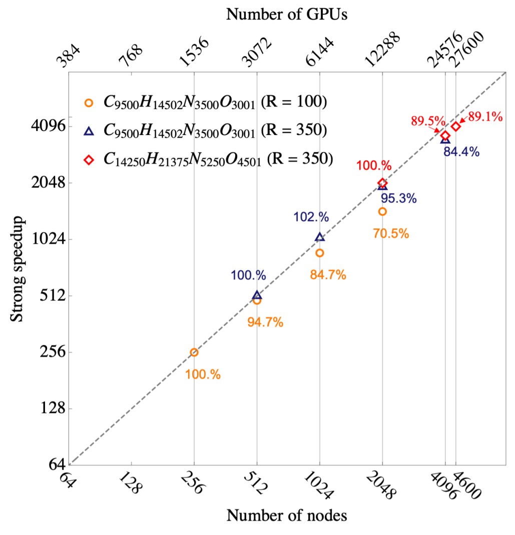


Figure 7: Log-log scale strong scaling speedup of SCF + RI-MP2 calculations obtained on the Summit supercomputer. Percentages indicate parallel efficiencies.

We developed a new fragmentation-based SCF-Hartree-Fock (HF) + RI-MP2 algorithm for many-GPU architectures. The new algorithm implemented a number of innovations compared to the state of the art. Among these are:

- Use of molecular fragmentation to enhance parallel distributed scalability and reduce the scaling of the algorithm from quintic to linear with system size.
- Use of a multi-layer dynamic balancing scheme that allows us to achieve high parallel efficiency across computational nodes, across GPUs on the same node and across multiple GPU threads.
- More efficient GPU kernels for the bottleneck of the HF algorithm, the Fock build, which computes integrals and postprocesses them on-the-fly for a fast Fock matrix formation completely on GPU.
- New RI-MP2 algorithm and implementation where, for the first time, the two- and three center integrals are evaluated directly on the GPU using kernels which are specifically tuned for the GPU architecture in use.
- The evaluation of the various MP2 tensor intermediates is performed with a double-stream algorithm designed to hide the latency of host to GPU data transfer, thereby maximizing GPU throughput.

Benchmarks show that the new code demonstrates remarkable speedups with respect to existing SCF and RI-MP2 codes, and excellent performance on Summit. Strong scaling calculations were performed on the Summit supercomputer on linear chains of amino acids, using 6-31G∗ as the primary basis set and cc-pVDZ-RIFIT as the auxiliary basis set.

Two systems were selected for benchmarking the strong speedup: the first with 500 Asp-Ala-Hys-Lys units ($C_{9500}H_{14502}N_{3500}O_{3001}$) leading to calculations involving $\sim$ 30,000 atoms and $\sim$ 120,000 electrons, the second with 750 Asp-Ala-Hys-Lys units ($C_{14250}H_{21375}N_{5250}O_{4501}$), over 45,000 atoms and over 180,000 electrons. Using the 6-31G∗/cc-pVDZ-RIFIT basis set combination, the first system included 312,525 primary basis functions and 1,273,596 auxiliary basis functions; the second system included 464,293 primary basis functions and 1,896,846 auxiliary basis functions, a staggering figure. For the first system dimer screening distances of $R = 100$ and $R = 350$ au were chosen to test the accuracy and the parallel scalability of the calculations. For the (Asp-Ala-Hys-Lys)750 system the dimer screening distance was fixed to $R = 350$ au, ensuring accurate results. A graph demonstrating the strong-scaling results is shown in Fig. 7.

Strong-scaling calculations for the (Asp-Ala-Hys-Lys)500 system with $R = 100$ were performed on up to almost half of the Summit machine. For this system, the implementation shows good scaling up to 1024 nodes, where it achieves an 84.7% parallel efficiency, and a lower 70.5% parallel efficiency on 2048 nodes.

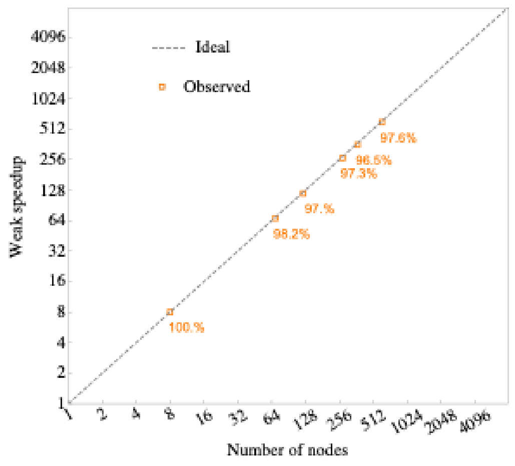


Figure 8: Log-log scale weak speedup obtained on the Summit supercomputer. Percentages indicate parallel efficiencies.

The performance deterioration for this calculation arises, not because of increased communication between the MPI processes (as corroborated by the excellent weak scaling results in Fig. 8), but because the number of nodes becomes too large compared to the number of fragment pairs in the chosen molecular system. In fact, on 2048 nodes each node receives only between 10 and 14 fragments, and the parallel workload becomes easily unbalanced.

Increasing the dimer-screening cutoff distance to $R = 350$ yields an improved strong scaling for the (Asp-Ala-Hys-Lys)500, and enabled us to test the convergence in the final energy of this system. Results are shown in Fig. 7 on up to 4096 nodes, with a good parallel efficiency of 84.4% on the largest node count. The improved scaling can be attributed to the beneficial effect on the algorithm's parallel workload balance yielded by the increased number of fragment dimers included in the calculation. Strong scaling calculation for the (Asp-Ala-Hys-Lys)750 system were conducted using up to 4600 nodes (99.8% of the whole Summit supercomputer) and 27,600 V100 GPUs. As shown in Fig. 7 the algorithm shows very good strong scaling, achieving parallel efficiencies of 89.5% and of 89.1% on 4096 and 4600 nodes, respectively.

Table 13 shows wall times for the (Asp-Ala-Hys-Lys)500 and (Asp-Ala-Hys-Lys)750 calculations with respect to the number of Summit nodes used. The (Asp-Ala-Hys-Lys)500 calculation using 2048 nodes on Summit ran in 6 minutes. The (Asp-Ala-Hys-Lys)750 calculation using 4600 nodes ran in about 11 minutes, and was by far the largest electronic structure calculation with quantitative accuracy ever performed.

Figure 8 shows the weak scaling of the SCF+RI-MP2 code from 8 to 612 nodes on Summit. In order to obtain the weak scaling data we timed the RI-MP2 code for calculations on water cluster systems using the 6-31G∗/cc- pVDZ-RIFIT basis sets with an increasing number of equal size fragments. For these calculations, we use a second order MBE considering all fragment dimers (no screening). Therefore, the total computational workload is proportional to the square of the number of basis functions; thus, increasing the number of nodes as the square of the system size will result in an approximately constant workload across nodes.

As shown in Fig. 8, the RI-MP2 code achieves linear weak scaling across the whole range of nodes considered, which varies between 8 and 612. The largest molecular system, with 2608 water molecules, reaches a weak speedup of $\sim 597\times$ on 612 nodes, which is also the largest node count.

HIP/Spock

The originally C++/CUDA-developed Self-Consistent Field (SCF) Restricted Hartree-Fock (RHF) algorithm was ported to the HIP programming model for AMD GPU execution. The initial porting was done using the hipify-perl tool on most of the files that contained CUDA directives, files and headers.

The SCF-RHF program contains a series of GPU accelerated routines that can be divided into:

- Dense linear algebra
- Quantum chemistry specific GPU kernels

Table 13: Wall times (in seconds) of the SCF+RI-MP2 calculations for the (Asp-Ala-Hys-Lys)500 = $C_{9500}H_{14502}N_{3500}O_{3001}$ and the (Asp-Ala-Hys-Lys)750 = $C_{14250}H_{21375}N_{5250}O_{4501}$ systems with respect to number of nodes used on Summit.

Molecule	N	N_X	R	Number of nodes	Wall time (s)
$C_{9500}H_{14502}H_{3500}O_{3001}$	312,525	1,273,596	100	256	2015
				512	1064
				1024	594.5
				2048	357.5
$C_{9500}H_{14502}H_{3500}O_{3001}$	312,525	1,273,596	350	512	3537.7
				1024	1734.5
				2048	928
				4096	524.2
$C_{14250}H_{21375}H_{5250}O_{4501}$	464,293	1,896,846	350	2048	1354.3
				4096	756.3
				4600	676.5

– One electron integrals (recursive approach)
– Two electron integrals (recursive approach)
– Digestion into Fock matrix

The dense linear algebra kernels depend on a BLAS library for their GPU execution, originally cuBLAS and cuSolver. The routines used are:

- Daxpy
- Dgemm
- Dscal
- Dsyev (cuSolver)

The hipify-perl tool was able to translate to hipBlas every single code for Daxpy, Dgemm and Dscal, however the Dseyv eigenvector solver is part of rocsolver and had to be translated by hand.

The kernels for the evaluation of the two electron integrals and their further digestion into the Fock matrix were originally developed using a Mathematica based code generator. The file for the evaluation of these routines contains around 50k lines of code for d functions and 2 million lines for evaluating f functions. The generator was altered to create a HIP version of the kernels. The build system was finally modified to accommodate a CUDA and HIP build system. Additionally, a HIP with CUDA backend compilation is possible under the current CMake.

So far, the RI-MP2 code has been "hipified" using the hipify-perl tool. The RI-MP2 code also has some cusolver eigensolver calls, which need to be translated by hand to either the rocblas/rocsolver and/or magma (not ideal). Based on the previous results of rocsolver, the proper route at the moment would be to implement the eigensolver routines using Magma, which has been proven to be working.

The SCF-RHF + RI-MP2 model requires Magma, which introduces a new dependency into the toolchain. Currently, Birch, the HPE system, does not have Magma installed, and using Spack to install it locally has not worked. This remains an issue for future testing and development.

DPC++

Previously, support for the generation of DPC++ kernels was added to the integral code generator used to generate the GPU ERI code used in LibCChem. The rest of the LibCChem code was translated either using the DPCT tool offered by Intel, or by hand in cases where DPCT did not work. However, the integral kernels generated by the integral code generator have been significantly altered in the past year. Most notably, the integral computation kernels have been combined with the integral digestion kernels, leading the code generator to generate combined integral computation plus digestion kernels. These alterations to the integral kernels have necessitated an update to the generation process of DPC++ kernels in the integral code generator. The updating of DPC++ kernel generation with the new integral kernels is currently a work in progress.

- Initial status of the code for AMD execution: Up and running on Tulip, Spock, and Birch.
- Initial status of the code for Intel execution: Should theoretically run on Crusher and Arcticus.

Problems with HIP

The following issues have been identified using HIP on AMD systems:

- The code uses an interpolation method to evaluate a core function needed by the one- and two- electron integrals; without it the code is useless. In order to evaluate the interpolation, a series of tables are copied over to the GPU for the kernels to interpolate on. Several kernels use a series of array definitions needed to access the arrays correctly. The array access needs the flag `fgpu-rdc` to be enabled. Enabling this flag creates the following problems:
  - Long linking time to produce executable (6-7 minutes) compared to 2-3 seconds on NVIDIA V100
  - Printf from inside a GPU kernel does not work—crashes the code with a segfault
- Performance of rocBlas and rocSolver when compared to cuBlas and cuSolver
  - Operations on a 105×105 matrix
    * rocBlas/rocSolver 0.5148 seconds—cuBlas/cuSolver 0.3441 seconds (transformation matrices)
    * rocBalas/rocSolver 0.637 seconds—cuBlas/cuSolver 0.007 seconds(DIIS)
  - Operations on a 1950×1950 matrix
    * rocBlas/rocSolver 50 minutes—cuBlas/cuSolver 0.4926 seconds (transformation matrices)
- Inconsistency between `maxrregcount` and `max_threads_per_block`
  - As per the HIP programming guide the hcc compiler does not support the `maxrregcount` option which allows the CUDA code to use a max number of registers defined by the user, dependent on the microarchitecture. HCC states that this is not very portable, which is true, however we believe in a good build system that will have the architecture information for the different generations of cards, including the number of registers they contain. We presently have such an implementation for NVIDIA.
  - The evaluation of the two electron integrals involves a large number of recursive intermediates which are stored in the register memory when possible; this leads to the code benefiting from a large register pool per thread. See comparison between 64 and 255 in Table 14.
  - As per a discussion on the Frontier COE the alternative is modifying the `max_threads_per_block` compile time variable or choosing launch bounds per kernel
    * `max_threads_per_block` option: The default value is 1024, the number was cut down by halves up to 256. 256 threads per block offered the best performance, as shown in Table 15, making the MI100 GPU perform at around 50% of a NVIDIA V100
    * Numbers below 256 caused failed-to-launch errors for kernels for larger chemical systems (¿500 basis functions) and wrong answers for smaller systems
  - Launch bounds option
    * Creating customized launch bounds for each kernel would be a gigantic—probably profiler driven—task. There are around 50-60 highly specialized kernels that drive the application, each completely different from the other. This would require profiling every single kernel for a large variety of systems to come up with the best launch bound configuration. Doable but highly impractical

Table 14: Fock build time (s) on an NVIDIA V100 GPU for 64 and 255 registers per thread.

RNA9	NBAS	64 Regs	255 Regs
STO3g	4363 25.4	8.9	
6-31G	7816 88.3	41	
6-31G∗	12412 119	91.5	

Table 15: Fock build time (s) on MI100 and V100 GPUs with a thread-block size of 256.

RNA9	NBAS	MI100	V100
STO3g	4363	12.7	8.9
6-31G	7816	76.9	41
6-31G∗	12412	234	91.5

∗ A silver bullet approach, such as the `maxrregcount` flag would be highly appreciated

Performance Data

In this section, the performance on the MI100 is assessed and compared against the V100. The heavy rocblas/rocsolver routines have been commented out, as it is the Fock build and two electron integral evaluation which are the true bottlenecks of the calculation. We believe that improvement in the software stack will allow the code to run at its full potential.

Four sets of calculations have been performed on the Spock system at OLCF that are shown in Figs. 9–12. All of the results were obtained using ROCM/4.3.0 as installed system wide on the Spock machine, linking with rocblas and rocsolver using the `max_threads_per_block` = 256 option for maximum performance of the kernels. From the results, it can be easily seen that the MI100 at the moment is producing a consistently lower performance when compared to a NVIDIA V100. Time to solution for the Fock build time is the limiting factor for any RHF calculation and will have a big impact on the time to converge the algorithm. The scaling against the V100 seems to be favorable, this is due to the slower time to solution by the MI100.

The code was then executed using the MBE fragmentation method used in previous studies on the Summit supercomputer and previous reviews. The code successfully ran using the 4 nodes available for to an ECP project on Spock. The test system was a set of 15 water molecules with each being an individual fragment. This leads to 120 total fragments, out of which 105 are fragment dimers. The timings are as follows:

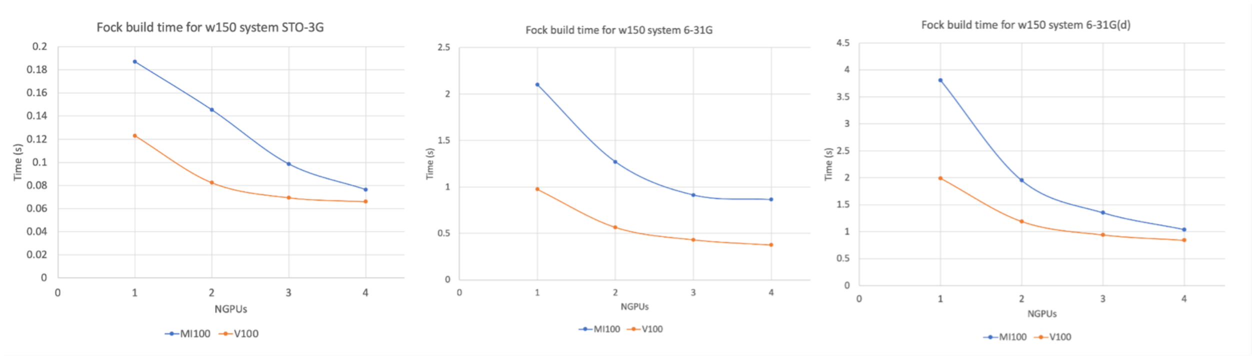


Figure 9: Fock build time for a system of 150 waters (450 atoms) using the STO-3G, 6-31G and 6-31G∗ basis sets using up to 4 GPUs.

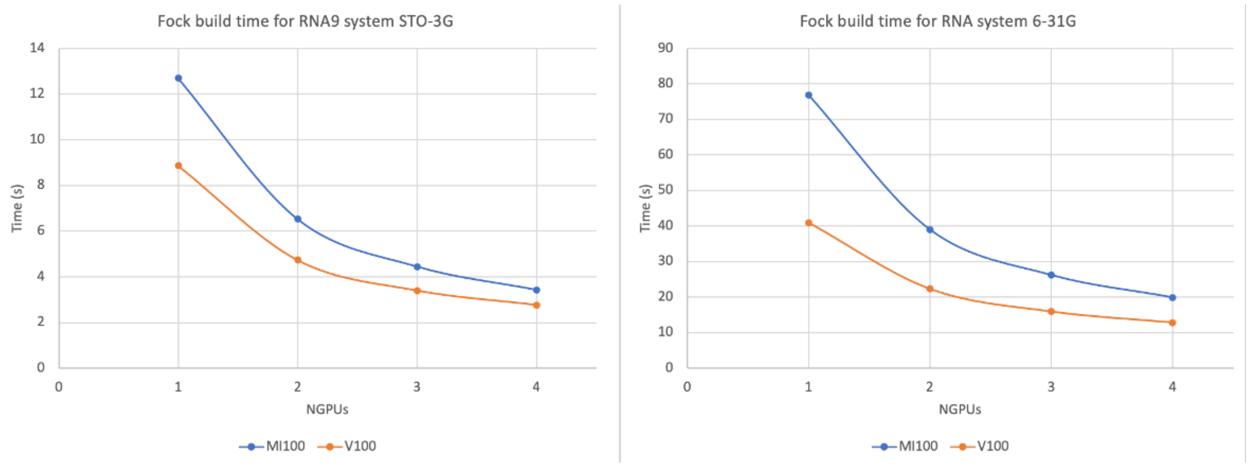


Figure 10: Fock build time for a system of 9 (ACGU) nitrogenated basis units (1115 atoms) using the STO-3G and 6-31G basis sets using up to 4 GPUs.

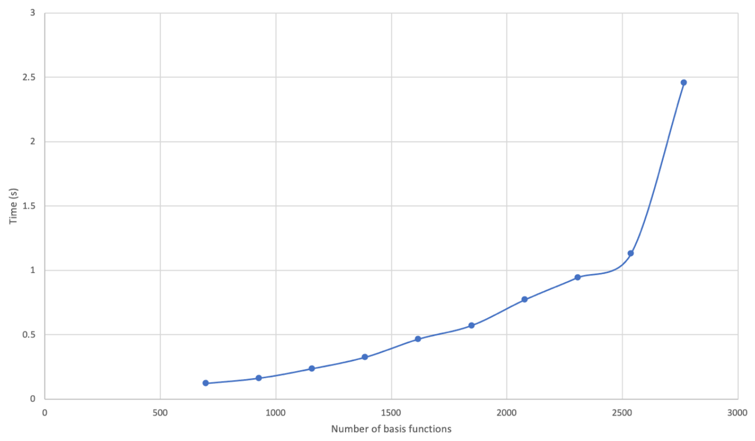


Figure 11: Fock build time for a system comprised of n Glycine units $n = 30$–120 in 10 unit increments using the STO-3G basis.

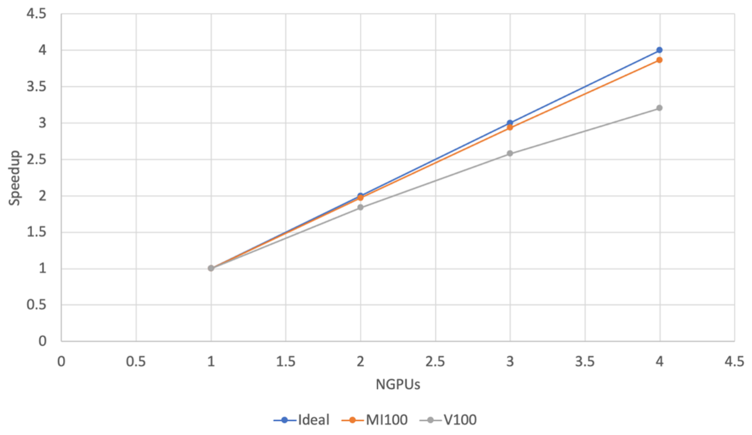


Figure 12: Scaling for Fock build time for RNA9 6-31G.

Table 16: Total time (sec) using 1 GPU or 6 GPUs for 6, 8, 16 and 32 water (w) molecules at HF/6-31G level of theory using the rotated axis sp code.

HF/6-31G	6 GPU	6w	8w	16w	32w
Local	atomic	2.63	5.85	19.4	90.11
Local	reduction	2.64	3.41	10.69	error*
private	atomic	2.13	3.46	9.05	36.04
private	reduction	2.88	2.95	8.72	error*

*an illegal memory access was encountered.

Nodes	Time (s)
1	30
2	25
4	17.7

Finally, other versions of ROCM were tested on the Birch test system at HPE. ROCM 4.2.0, 4.3.1, 4.4.0, and 4.5.0 were tested. The lack of performance of the rocblas and rocsolver routines were seen.

An option to circumvent the poor performance of rocsolver is to use MAGMA as the math library backend. An initial implementation shows a significant speedup versus the ROCM libraries. However, as of Nov 12th 2021, there are bugs to resolve regarding the MAGMA implementation for larger chemical systems.

As a final note, regarding the long linking times: AMD has reported back to the GAMESS ECP team discussing the origin of the problem. As suspected, the device relocatable code is to blame. Two suggestions were made:

- Put all the device calls into a single translation unit i.e. source file.
  - Good idea, simple fix but very messy from the software engineering

  perspective
- Wait for AMD engineers to improve this
  - Long term, but probably better

Offloading Electron Repulsion (ERI) MiniApp

Summit/Ascent

xlf xlf is the primary offloading compiler that we use for developing GPU code. We typically develop code on Summit/Ascent with xlf and make sure the code gives correct answers and then adapt it for other compiler/hardware systems. We now have MPI parallelization implemented for the miniERIs which partitions the outermost loop of the quartet loops into chunks. This allows the code to distribute the workload to multiple GPUs. Currently, the code maps 1 MPI rank to 1 GPU.

Array reduction is available for xlf 16 to replace the `omp atomic add`. In addition, there are quite a few intermediate quantities in the algorithm that are computed and used to calculate the final array we need. We investigated having such arrays either passed as an argument in the subroutine calling list and need to be declared OpenMP private or remain as a local intermediate quantity. Therefore, in total we have 4 combinations: local + atomic, local + reduction, private + atomic and private + reduction. It was found that, (1) changing from atomic add to array reduction always improved the time; (2) when using the same summation approach (either atomic add or reduction), having the intermediate quantities as OpenMP private variables and passing the argument list demonstrated better timing than having them as local variables. An "illegal memory access" error was encountered when going to larger water clusters. Results for these cases are shown in Table 16.

Table 17: The kernel time of the miniERIs using NVHPC for various sizes of water clusters using 6-MPI ranks.

waters	nshells	Time (s)
8	56	1.09
16	112	4.7
32	224	24.29
64	448	268.22

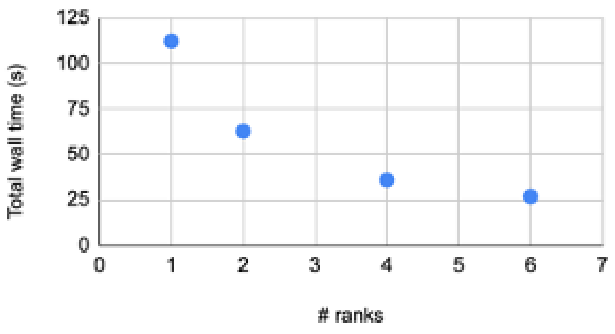


Figure 13: Scaling plot of the miniERIs for 32-water cluster at RHF/6-31G level using NVHPC SDK 21.9 compiler on Summit.

NVHPC SDK 21.9 The offloading code is mainly developed with the IBM compiler. There are some additional implementations required to make the code work with NVHPC. One of the main reasons is that NVHPC cannot handle allocatable arrays that well.

Here is an example of the issues that we experienced. Using the following compiler flags, we got a compilation error.

```
FC=mpif90
FFLAGS= -fast -Minline -mp=gpu -Minfo=all -i8
F_LKED= -fast -Minline -mp=gpu -Minfo=all -i8
```

If the `-i8` flag was removed, the code can pass compilation. However, a runtime error is encountered. The issues have been communicated to OLCF support and a bug report has been filed to NVIDIA. To resolve these issues, we have to do a redundancy of data mapping from host to device by using both `omp target enter data map` and `omp target data map`. Moreover, we found that having private arrays in the target region causes runtime errors when using multiple numbers of teams and threads. The solution is to move those arrays into the kernel locally. The performance of miniERIs has been tested for the 32-water cluster using NVHPC SDK 21.9 on Summit (see Table 17 and Fig. 13).

gcc 11 Currently, the code can be compiled with the flag

```
-fopenmp -foffload=nvptx-none -lm -lgfortran -Ofast -latomic -fdefault-integer-8
```

However, the runtime error of illegal memory access was encountered again. This error appears to be related to the atomic add, since if atomic add is commented out, the calculation could run to finish, of course with incorrect numerical results. Replacing the atomic add by the array reduction resulted in the same illegal memory access error. This has been identified and reported to the gcc team during the October OpenMP Hackathon.

Spock

CCE 12 The CCE compiler had difficulty handling too many OpenMP private variables, which resulted in a linking error as shown below:

```
error: <unknown>:0:0: in function gpu_ompmod_twoei_jk$gpu_ompmod_$ck_L214_1
void (i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64,
i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64,
i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64, i64,
i64, double, i32, i32, i32, i32, i32): unsupported dynamic alloca
```

During the October OpenMP Hackathon, we have attempted several things after playing around with some reproducer code that resulted in the same linking error. Attempts included (1) changing some variables to assumed-size array (2) remove `declare target` for some unused common blocks, (3) move some arrays from OpenMP private arrays to local arrays. While steps (1) and (2) seem to make the linking error message shorter, step (3) is the key to resolving this linking error. Unfortunately, a compilation error has now occurred. miniERIs can compile, and the test ran to completion when removing all the flags, resulting in incorrect numerical results. Simply adding the `-h omp` flag is already problematic when compiling.

Note that all test suites for Fortran OpenMP offloads created by the SOLLVE team (https://crpl.cis.udel.edu/ompvvsollve) failed the with CCE compiler on Tulip. However, we will continue to improve the performance of our miniERIs and test the code when the new CCE version is available.

Off-Loading RI-CC on Spock

All parts of a CCSD(T) calculation consist of vectorized matrix operations, in particular, matrix-matrrix operations. Therefore, the RI-CC code is heavily dependent on BLAS libraries.

On AMD GPUs, we attempted to combine OpenMP off-loading with HIPBLAS. This was not successful. The matrices mapped to the device via OpenMP target constructs (`omp target enter data`) are not recognized by the HIPLBLAS routines, and null results are obtained after matrix-matrix multiplications (even though the expected speedup relative to execution is observed). As a solution, we are trying to use HIP routines to allocate memory and perform BLAS operations instead of using OpenMP offloading.

3.4 EXAALT

3.4.1 KPP Verification Plan

The fusion challenge problem that forms the base goal of the EXAALT project requires a dramatic extension of the reach of large-size, long-time molecular dynamics simulations. The team aims to simulate the evolution of a tungsten first-wall in conditions typical of fusion reactor operation. The primary target is to simulate a 105-atom system with a quantum-trained SNAP potential. This second challenge problem is used to define the threshold goal and hence a KPP that will quantify the team's progress. The challenge problem requirements are listed in Table 18.

Verification Simulations

Facility Requirements

We only require MPI and Kokkos, both of which are standard or ECP-supported software technologies. No additional facility support is expected for demonstration calculation, but testing of experimental features might benefit from facility involvement. For example, experimental neural network extensions of SNAP would require supported python machine learning (ML) modules, such as pytorch.

Input Description and Execution

We will create a configuration representing a damaged tungsten surface that is representative of what would be expected on the first wall of a fusion reactor. The configuration will contain around 105 atoms. The initial state will be partitioned into sub-domains for use with the Sub-Lattice ParSplice implementation, yielding on the order of 50–100 sub-domains. The rest of the input will simply specify run conditions, such as temperature ($T = 800\,\text{K}$), potential used (SNAP with 205 descriptors), and parameters of the ParSplice calculation (buffer sizes, molecular dynamics (MD) time step, etc.); these will be set following conventional guidelines available from the open literature. At this point, we do not foresee using advanced SNAP potentials

Table 18: EXAALT challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	This problem consists of investigating the evolution of a tungsten surface under conditions relevant to exposure to a fusion plasma using atomistic simulations. The main physics of interest are the annealing mechanisms and characteristic timescales. This problem will be modeled by using a SNAP representation of tungsten. This will access the intermediate-accuracy/long-time/intermediate-size regime.
Numerical approach and associated models	Parallel trajectory splicing—parallel molecular dynamics.
Simulation details	The reference simulation consists of a sublattice ParSplice simulation of a 10^5 atom system with a damaged tungsten surface typical of plasma-exposed conditions. The simulation will be carried out at $T = 800\,\mathrm{K}$, which is a fusion-relevant temperature, by using a SNAP representation of tungsten. The baseline FOM corresponds to a SNAP parameterization that uses 205 bispectrum components, or 205 descriptors of local atomic environments. A greater number of bispectrum components gives higher accuracy, and this number of components is consistent with the high accuracy desired for the final science challenge problem calculations. Improvements in the SNAP form developed under this ECP subproject, such as the new quadratic form or neural network extensions, might allow the team to ultimately achieve this same accuracy with fewer bispectrum components with reduced cost. Accuracy is quantified as the average error in predicted forces relative to a large database of DFT calculations. In the final benchmark calculation used for the Fusion FOM, either the baseline SNAP form or an improved SNAP form with accuracy equivalent to the one used for the baseline calculation will be used.
Demonstration calculation requirements	The team anticipates requiring only a few runs at scale to demonstrate this capability. To obtain an accurate performance benchmark on the order of 50,000 time steps on each worker ($\sim 50\,\mathrm{ps}$ of simulation time) are needed.

for the demonstration calculation itself, but these might be tested at scale upon successful demonstration of the baseline simulation. Each demonstration simulation will be relatively short, on the order of 1 to 2 hours of runtime. We will gradually increase the scale of the simulation until reaching full-machine runs.

This would correspond to the generation of about 50,000 time steps on each worker, consistent with the target identified in the "Demonstration calculation requirements" section.

Simulation Artifacts

The simulation produces two kinds of artifacts: sequence of atomic configurations of the system and performance measurements. The performance file (called times.out) directly reports the number of "local" trajectory blocks (corresponding to evolving individual subdomains in time) that were received at the master process as a function of time. Multiplying the number of blocks by the number of time steps per block and by the number of atoms per local domain and dividing by the wall-clock time will yield the FOM measured in atom*time step/wall-clock second. We will provide a simple script that will perform that conversion and produce the final KPP value.

Verification of KPP-1 Threshold

The final FOM will be obtained by dividing the exascale KPP value (obtained using the procedure described above) by the reference value obtained on Mira. A FOM value greater than 50 will indicate success.

The minimum criteria are fairly simple in our case, consisting of the specification of the system size and energy model used to carry out the simulations. Both these characteristics can be assessed directly in the input files described above. Specific documentation will be provided to indicate how these can be verified in the input files.

The physics produced by the code will also be validated on a simple test problem of a surface containing isolated tungsten adatoms. These will be simulated in serial at multiple temperatures. The hopping rate of this adatom will be measured as a function of temperature and compared to predictions from transition state theory using static transition barriers. The serial code will also be validated for energy conservation and consistency between energy and gradients. This will establish a set of reference values for kinetic properties. A large test system composed of many isolated adatoms will then be created and simulated in the Sub-Lattice ParSplice code, again at various temperatures. The average adatom hopping rate will again be measured and compared to the reference implementation. Statistical significance analysis based on Poisson statistics will be provided to quantify the agreement between the reference and ParSplice simulations.

3.4.2 Early Hardware Status

Optimization of the key computational kernels has continued in FY21. The current performance on V100 is at 132 katom step s^{-1}, an improvement of close to 26× with respect to the baseline code. Note that highlights code progression on identical hardware. This is an approximately 20% increase from 110.7 katom step s^{-1} from the FY20 review. Code performance has been further improved for large atom counts, which does not impact performance in conditions relevant to the EXAALT KPP. The code is now highly optimized, especially for NVIDIA hardware such as the V100 on Summit and A100 on Perlmutter. Efforts during FY21 have shifted from optimization to porting to exascale testbeds.

We note that optimization of the SNAP-Kokkos kernels has been a very successful example of collaboration both within and outside of ECP. Stan Moore (CoPA) led the development of the overall LAMMPS/Kokkos code. Aidan Thompson and Nick Lubbers (EXAALT) led the algorithmic redesign of the force call, Rahul Gayatri and Neil Mehta (NERSC, through the NESAP program with EXAALT) led the development of the Kokkos OpenMPTarget backend and the deployment and porting of LAMMPS and TestSNAP on the testbeds. Chris Knight and Yasi Ghadar (ALCF) assisted in the port of TestSNAP and LAMMPS on Iris and Arcticus. Daniel Ardnt (ORNL) led the development of the Kokkos SYCL backend and helped tune TestSNAP on Arcticus.

Status on NVIDIA Hardware

Both the proxy-app and the production codes compile and run correctly on V100 and A100. Performance on these systems has proven quite high, with peak throughputs of 132 katom step s^{-1} on V100 (see Fig. 14) and 174 katom step s^{-1} on A100. The EXAALT suite has been successfully running in production on

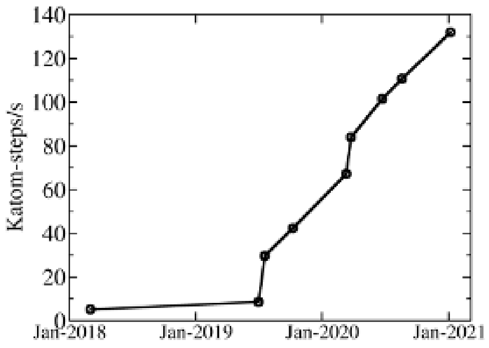


Figure 14: Progression of the performance of the SNAP calculation w.r.t. the pre-ECP baseline code as measured on a single V100 GPU. The performance of the baseline code was 5.1 katom step s^{-1}, while the current production code delivers 132 katom step s^{-1}, for an increase of 25.88×.

Table 19: Current status of the TestSNAP proxy-app and LAMMPS production codes. Performance is measured in katom step s^{-1}.

Hardware (Kokkos backend)	TestSNAP	LAAMPS	Slowdown vs. V100 (LAAMPS)
Summit V100 (CUDA)	124	132	1.0×
Spock MI100 (HIP)	63	55	2.4×
Perlmutter A100 (CUDA)	179	174	1.32×

A100/Perlmutter up to 1/3 scale (2048 GPUs). Full machine runs are upcoming. The code is very stable and runs as expected.

Status of AMD hardware

Both the proxy-app and the production codes compile and run correctly on MI60 and MI100. Recent focus has been on investigating performance on the Spock system using the HIP Kokkos backend running on MI100. Performance so far has been lower than expected, with a measured peak throughput of 55 katom step s^{-1}. Current investigation points to the limited L1 cache on MI100 (96 kB per SM on V100 vs 16 kB per SM on MI100). This 6× reduction in L1 cache size appears to cause significant slowdown. Indeed, artificially limiting the L1 cache available on V100 also leads to a significant decrease in performance. As will be discussed below, current efforts to fully understand, characterize, and potentially mitigate this issue are slowed down by the lack of adequate profiling tools. Performance on MI100 has modestly increased by about 5% since the previous benchmarks. In spite of the lower-than-expected performance, the current code provides correct answers and would deliver a more than 700× increase in FOM with respect to the reference value on Mira, assuming an MI100-based exascale machine.

A comparison of the performance of TestSNAP and LAAMPS on NVIDIA and AMD hardware is shown in Table 19.

3.5 ExaAM

3.5.1 KPP Verification Plan

Verification Simulations

The Exascale Additive Manufacturing project (ExaAM) is developing a suite of exascale-ready computational tools (framework) to assess the manufacturability by additive manufacturing (AM) methods and performance of designed components. Because of the sensitivity of material microstructure to manufacture process and local material property to microstructure, this goal requires process simulation at the fidelity of the microstructure. Performance and qualifiability are assessed using the resulting process-aware material

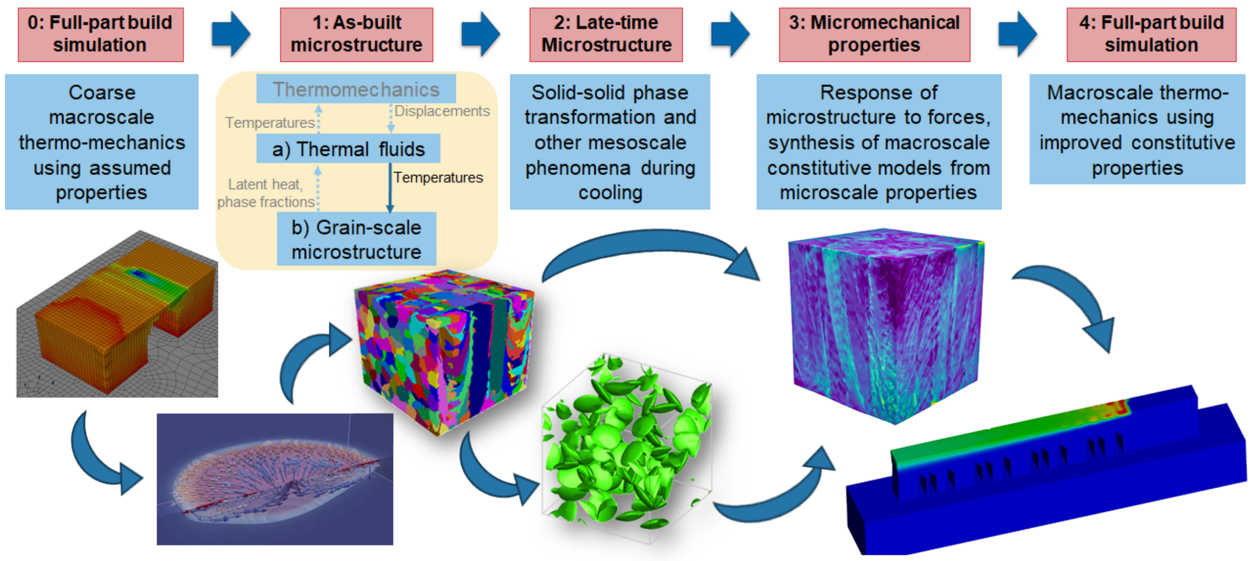

0: Full-part build simulation
1: As-built microstructure
2: Late-time Microstructure
3: Micromechanical properties
4: Full-part build simulation
Coarse macroscale thermo-mechanics using assumed properties
Thermomechanics
Temperatures
Displacements
a) Thermal fluids
Latent heat, phase fractions
Temperatures
b) Grain-scale microstructure
Solid-solid phase transformation and other mesoscale phenomena during cooling
Response of microstructure to forces, synthesis of macroscale constitutive models from microscale properties
Macroscale thermo-mechanics using improved constitutive properties

Figure 15: The five stages of the ExaAM simulation workflow

model in macroscopic finite-element simulation of component behavior.

The required fidelity of both the microstructure and performance simulations are assessed via validation against experimental observations for each stage of the ExaAM workflow (Fig. 15 and Table 20).

This capability will be demonstrated by simulating the Inconel 625 (IN625) build of the complex bridge structure developed for the 2018 National Institute of Standards and Technology (NIST) AM-Bench Conference known as AMB2018-01 [13]. AMB2018-01 is a carefully characterized component build from which further microstructure observations are being performed to both guide the model development and validate the simulations.

The threshold simulation will be performed at the location at which measurements were performed, 2.5 mm above the base plate, for one of the thick legs. Since the alloy selected is IN625, which does not exhibit significant microstructure changes due to solid-solid phase transformation and precipitate formation during the build process, Stage 2 can be neglected from the threshold problem. However, Stage 2 would be required for other materials and other processing conditions (late-time annealing), such as IN718 or Haynes282, and other processing conditions (late-time annealing), so stage 2 appears as a stretch science goal.

The challenge problem details for stages 1 and 3 are listed in the tables below. The following caveats apply to the challenge problem:

- This estimate is for the threshold challenge problem only. The actual goal would be to predict microstructure and properties everywhere throughout AMB2018-01, which would require significantly more computational resources.
- This estimate includes a significant amount of flexibility (e.g., number of layers in Stage 1, number of representative volume elements (RVEs) in Stage 3), allowing adjustment based on accuracy needs, better-than-expected performance, or lower-than-expected performance.
- Memory is not anticipated to be a limiting factor to run the challenge problem.

Threshold Challenge Problem Metrics

Successful completion of the threshold challenge problem will be determined by assessing ExaAM predictions of micromechanical properties and the strain field in the as-built IN625 AMB2018-01 with respect to (1) comparison to experimental values, and (2) improvement over baseline values at the start of ExaAM (when available). Table 21 gives a summary of these metrics. Speedup and scaling for the two most computationally intensive components required for the threshold challenge problem, ExaCA and ExaConstit,

Table 20: Physics components and experimental observations associated with each stage in the ExaAM workflow.

Stage	Description	Component(s)	Experimental Observations
0	Approximate full-part build simulation	Diablo	Elastic strain field (x-ray and neutron), max/min values and locations, integrated difference, temperature (thermocouple and IR)
1a	Melt pool	TruchasPBF, OpenFOAM, PicassoMPM	Melt pool width, depth, and morphology, temperature (IR), absorptivity, interface velocity (APS)
1b	As-built microstructure	ExaCA, CAFlow (stretch), AMPE (stretch), Tusas (stretch)	Grain size, distribution and morphology, texture (2D and 3D Orientational Imaging Microscopy (OIM) and Orientation Distribution Function (ODF))
2	Late-time microstructure	MEUMAPPS-SS (stretch)	*Not required for threshold challenge problem due to material used (IN625)*
3	Material properties	ExaConstit	Micromechanical stress-strain response
4	Locally-accurate, process-informed full-part build simulation	Diablo	Elastic strain field (x-ray and neutron), max/min values and locations, integrated difference, temperature (thermocouple and IR)

Table 21: Threshold challenge problem physics metrics

Quantity of Interest	Experimental Difference	Improvement over Baseline	Stage
stress at 30% strain	5%	N/A	3
stress at 0.2% offset yield stress	5%	N/A	3
maximum value of elastic strain at experimental locations	10%	20%	4
minimum value of elastic strain at experimental locations	10%	20%	4
integrated elastic strain	5%	10%	4

Table 22: Threshold challenge problem exascale utilization metrics

Component	Speedup w.r.t. Baseline	Speedup (GPU vs. CPU)
ExaCA	10x	10x
ExaConstit	20x	15x

will demonstrate usage of exascale resources as shown in Table 22. Performance of other components are considered stretch goals.

Simulation Artifacts

Artifacts for the threshold challenge problem and stretch science simulations are shown in Tables 23 and 24 for physics output files and exascale utilization. The physics output files map directly to the challenge problem and document its execution on the exascale system. Scaling results will be documented to illustrate the utilization components in the ExaAM workflow designed to execute primarily on the CPUs (OpenFOAM and Diablo). Other codes (ExaCA, ExaConstit, and PicassoMPM) are designed to exploit the GPU accelerators and speedup relative to the CPUs will be documented to demonstrate full exascale system utilization. GitHub links and internal ORNL code repository links are provided in order to allow direct access to the software components.

Results to date on Summit and pre-Exascale systems are included in the FY21 milestone reports:

- FY21 Q2 Milestone Report (Capability)
- FY21 Q4 Milestone Report

The FY21 Milestone Reports document results for both physics and computational aspects of each of the ExaAM component codes running problems on the progression to the threshold challenge problem. Timings are listed for pre-exascale systems and outputs files are available showing the physics.

3.5.2 Early Hardware Status

Spock, Crusher, and/or Arcticus early access results for ExaAM component codes are shown in Table 25 for challenge problem components and Table 26 for stretch science components.

3.6 QMCPACK

3.6.1 KPP Verification Plan

As detailed in earlier reports and milestones, the challenge problem is to calculate the cohesive energy of a large supercell of nickel oxide (NiO) by using QMCPACK and diffusion quantum Monte Carlo (DMC) to an accuracy of 0.010 eV per NiO formula unit at capability scale in a reasonable and scientifically productive amount of wallclock time (e.g., < 1 day). The team anticipates running a minimum 256 atom supercell up to a 1024 atom supercell, depending on the available accelerator memory and efficiency of the QMCPACK code at the time the simulations are run. The problem is as specified in the team's original proposal and subsequently refined to include the formal power law scaling of the method with system size to allow different system sizes to be run. The archived reference FOM data with a 128 atom supercell was obtained on Titan using the GPU accelerators and the CUDA implementation available in QMCPACK at that time. Larger supercells were not practical on Titan due to the 6 GB GPU memory.

NiO was selected as emblematic of the science challenges involving the complex physics of transition metal oxides. This classic Mott insulator (more accurately a charge transfer insulator) defies non-empirical predictions by other methods. Success for the NiO problem will indicate that a high and productive rate of computational work could be achieved for other challenging materials, including those with strong electronic correlations, novel magnetic states, and a host of novel quantum phases.

Verification Simulations

Table 23: ExaAM threshold challenge problem simulation artifacts.

Component	Physics	Exascale Utilization
Stage 1a: OpenFOAM	For each layer: thermal history files used for input to ExaCA (x, y, z, solidification start time, cooling rate) · L8 (small): 20 files, 5 Mb per file · L7 (large): 20 files, 50 Mb per file	· Run time (CPU) · scaling https://code.ornl.gov/openfoam-am/
Stage 1a: TruchasPBF	· thermal history · G/V/solidification start/stop times output for future ML acceleration use	https://github.com/LANL/Truchas
Stage 1a: PicassoMPM	· thermal history · phase history · velocity history · liquid-solid interface topology · liquid-vapor interface topology · absorption from ray tracing model	· GPU/CPU speedup · Scaling studies https://github.com/picassodev/picasso
Stage 1b: ExaCA	For each of the two legs: · Paraview plots of grain misorientation relative to the build direction in the solidified legs (0.5-5 GB, depending on the leg simulated) · Mean grain area as a function of build height in a representative region away from the substrate/walls (3 kB) · Mean weighted grain area as a function of build height in a representative region away from the substrate/walls (3 kB) · Output file(s) of x, y, z, and Grain ID passed to ExaConstit, where a representative $0.5\,\text{mm} \times 0.5\,\text{mm} \times 0.5\,\text{mm}$ region (at 2.5 micron resolution) would be 250 MB (multiple representative regions may be passed to ExaConstit)	· GPU performance improvement demonstration over CPU (5 kB) · Scaling studies on GPU and CPU https://github.com/LLNL/ExaCA
Stage 3: ExaConstit	Per RVE, loading condition, and temperature: The following quantities are needed to compute macroscopic yield surface parameters for Stage 4 · Yield surface parameters per RVE (∼1 KB) · *Volume average values per time step (∼10s KBs per file)* · Cauchy stress tensor · Plastic deformation tensor · *Volume values per time step (∼10s KBs per file)* · Plastic work Optional output: · Hydrostatic stress, lattice orientation, Cauchy stress, plastic slip rates, and many others related to crystal mechanics field variables per element (∼1s-10s GBs per time step)	· GPU performance improvement demonstration over CPU (5 kB) · Scaling studies on GPU and CPU · Caliper data showing runtime % per scopes of code https://github.com/LLNL/ExaConstit
Stage 4: Diablo	· Strain field throughout part	· Run time (CPU) · scaling

Table 24: ExaAM stretch science simulation artifacts.

Component	Physics	Exascale Utilization
Stage 1b: AMPE	· time-steps with diagnostics (solid fraction, min/max temperature,...) and timings · visualization files (1 file/task) with fields information (phase, composition, grain orientations, temperature) over time of simulation for post-run analysis with VisIt · restart files	· GPU/CPU speedup · Scaling studies https://github.com/LLNL/AMPE
Stage 1b: Tusas	· 2D melt pool microstructure	· large scale Summit GPU utilization · preparation for INCITE allocation https://github.com/chrisknewman/tusas
Stage 1b: CA Flow	· run time for each function on each MPI task · grain location, orientation and total volume · visualization files that include temperature, grain number, grain orientation, solute concentration, solid fraction · for control volume $300 \times 300 \times 300$ domain, output file size is ~ 120 MB per time step · for the single track case, output file size is 300 GB per time step.	· GPU/CPU speedup https://code.ornl.gov/langyuan/iamcool
Stage 2: MEUMAPPS-SS	· incubation time for nucleation · phase, variant selection · phase, variant morphology · phase composition · gradient precipitation morphology as function of initial concentration, concentration gradient · evolution of solidification microstructure obtained from AMPE	· GPU/CPU speedup using different FFT packages · Scaling studies https://github.com/ORNL/meumapps_ss

Table 25: ExaAM challenge problem component codes on ECP early access machines.

Component	Comments
Stage 1a: OpenFOAM	Spock: · Builds/runs correctly on AMD CPU with GNU compiler · OpenFOAM does not have any plans for GPU support Results for threshold problems · Linear scaling up to ∼10,000 cells per CPU on a single node · Local mesh refinement (LMR) was applied to reduce CPU requirement · Target cells per LMR zone was 1.6 m · L8 (thin leg) using 128 CPUs on 2 nodes: 680 seconds per layer, **total: 3.78 hrs** · L7 (thick leg) using 128 CPUs on 2 nodes: 7970 seconds per layer, **total 17.71 hrs** Issues running several MPI instances in parallel using `srun &` calls. · Need SLURM to schedule MPI jobs for time parallelization (improves computational scaling of threshold problem)
Stage 1a: PicassoMPM	Using Kokkos HIP and SYCL backends for Spock and Arcticus testing (no significant OpenMPTarget work) Builds/runs correctly on AMD GPU · Currently 2-4x slower on Spock than Summit · Primary slowdown due to particle-based kernels heavily using atomic operations · Larger system sizes are more performant (Spock relative to Summit) All dependencies build/run correctly on Intel GPU · Performance testing to be done on Arcticus (HIP testing prioritized to date)
Stage 1b: ExaCA	Using Kokkos HIP and SYCL backends for Spock and Arcticus testing (no significant OpenMPTarget work) Builds/runs correctly on AMD GPU · Currently ∼2x slower on Spock than on Summit · Slowdown likely due to reliance on atomic operations for main computational kernel · Slowdown is more significant for smaller systems and with increased GPUs (strong scaling) (Spock relative to Summit) Builds/runs correctly on Intel GPU · Performance testing to be done on Arcticus (HIP testing prioritized to date)
Stage 3: ExaConstit	CPU port to Spock using GNU compiler passes all tests ExaCMech ported over to HIP with a number of compiler workarounds in use · Submitted a bug report to OLCF about an internal compiler error with ROCm Ported code is ∼3x slower on Spock than on Summit when running ExaCMech miniapp · Recently discovered that ExaCMech was inadvertently relying on ATS features of the CORAL1 machines to move certain class member objects onto the device. The HIP copies this data over every time the RAJA `forall` loops are called, while the CUDA runs only do it once. We know this results in roughly a 20% slowdown if we behave the same as HIP on the CORAL1 machines. We expect moving over to a `chai::managed_ptr` for these class member objects or potentially a unified memory approach should close the gap a bit more. · It is believed part of the slowdown is due to the ROCm compiler not optimizing the code as well as the NVCC compiler given the similar technical specs between the Nvidia V100s and the AMD MI100s. ExaConstit has not been ported to HIP yet · Should be a simpler process than ExaCMech as MFEM has already been ported

Table 26: ExaAM stretch science component codes on ECP early access machines.

Component	Comments
Stage 1b: AMPE	Spock · builds and runs on CPU (CRAY compiler) · compiler issues prevent building OpenMP offload code—expect (at least some) issues to be corrected in later LLVM releases
Stage 1b: Tusas	· Straightforward port to CPU/OpenMP (Kokkos) with GCC and LLVM; all regression tests pass · Some required Trilinos components fail to compile/may not run correctly on GPU/ROCm (Kokkos)1
Stage 3: MEUMAPPS-SS (Fortran)	Builds and runs correctly with both CCE and GNU compilers on Spock for CPUs Builds and runs correctly with CCE compilers and OpenMP offloading for GPUs · All computing loops have been offloaded to the GPUs, only the FFTs remain on CPUs · Efforts are underway to move FFTs to the GPUs via heFFTe · Offloaded loops run extremely well on Summit: 65x speedup for 6 GPUs vs 6 CPUs, 18x speedup for 6 GPUs vs 42 CPUs · Poor GPU performance on Spock with original code that runs well on Summit · Refactored code performs significantly better on Spock (as of Nov 18, 2021) Results for 168∧3 problem with 15 variants for 20 time evolution iterations on 4 MPI ranks (1 CPU and 1 GPU per MPI rank): · Summit: 730 seconds on CPU-only, 232 seconds on CPUs and GPUs (with original code) · Spock: 415 seconds on CPU-only, 198 seconds on CPUs and GPUs (with refactored code) Ongoing efforts to optimize the code further for Spock and to find a common code that is performant on both Summit and Spock
Stage 3: MEUMAPPS-SS (C++)	Builds and runs on Spock, with verified simulation results for single-node CPU, single-node GPU, and multi-node GPU test simulations · Currently requires workaround OLCFDEV-388 to compile · Preliminary performance testing shows significantly worse GPU simulation performance on Spock relative to Summit ($\sim$68x), for 168∧3 problem with 15 variants for 20 time evolution iterations: · Summit: 12 seconds (6 MPI ranks, 6 GPUs) · Spock: 813 seconds (4 MPI ranks, 4 GPUs)

The simulation used to compute and report the FOM will be obtained from a single job using a large fraction of each machine. The simulations will be run identically to the FOM reference run and its input wavefunctions, except for supercell size. The reference run specifies the construction of the input trial wavefunction and DMC time step. The DMC equilibration period is also fixed. The supercell size determines the total atom and electron count, with the formal numerical cost scales with the third power of electron count.

The exact run configuration will depend on details of the memory size and performance of the GPUs on Frontier and Aurora as well as the eventual performance of QMCPACK for different electron counts. The most likely run configuration is a 512 atom run of NiO using as many Markov chains as fit on each GPU and to use every single available node on the machine. Larger (1024) and smaller (256) atom supercells will be considered. Due to the use of batched algorithms, QMCPACK gains efficiency when more Markov chains ("walkers") are propagated on each GPU. However, because the FOM includes an equilibration phase, it may be more effective to use fewer walkers.

Due to the high scalability expected ($> 95\%$) compared to single node runs, the demonstration run will be chosen based largely on single node performance. Any problems with the MPI implementation and network will be scouted via, e.g., quarter-scale runs ahead of the full-scale demonstration run. The demonstration run is expected to require under one day on each entire machine. A repeat of the 128 atom calculation that was run on Titan for the FOM reference data will require well under one hour on the exascale machines.

To obtain the highest FOM above threshold, several attempts will be required for different problem sizes and to account for inevitable improvements in the system software as the exascale machines mature.

Simulation Artifacts

Before running the FOM, we will record the pass/fail status of all the standard QMCPACK tests available at the time. If there are failures, the relevance to the FOM problem will be analyzed and recorded. For example, a failure in the Gaussian basis set support is unlikely to be relevant, while failures in the carbon diamond DMC test cases would be considered to make the FOM run invalid since the code paths are common.

While we do not know what the final DMC energy of the computed NiO supercell should be, we do know that due to finite size effects it should be energetically close to the energy of smaller cells, such as for the 128 atom FOM reference data. For infinite supercells, the energy becomes constant on a per primitive cell basis. The variance of the energy should also approximately scale with the atom count. Large changes in either the energy or variance from these expectations likely indicate a bug and invalid FOM.

The standard output files and analysis tools from QMCPACK will be utilized to compute the FOM. The standard output of the application includes timing information for the diffusion quantum Monte Carlo section, while the step-by-step progression of the DMC algorithm is recorded in a *.dmc.dat file. This is a text log that includes the average walker energies at each time step of the calculation. The error bar is computed from this data. Output will be analyzed using the standard "qmca" python analysis tool included with QMCPACK and following the community-standard procedures used published work, and identical to the procedures used to compute the reference FOM error bar and timings.

Since all output files produced by QMCPACK are small (MB), we intend to archive all the run outputs. The statistical analysis from the "qmca" tool will also be recorded. If sufficient storage is provided (multiple GB), all the source codes and problem inputs will be archived as well as the final outputs.

3.6.2 Early Hardware Status

We begin by evaluating the performance of the HIP port of the legacy CUDA implementation in QMCPACK. This implementation was used to establish the reference performance data for the project FOM. While CUDA is the native programming model on NVIDIA GPUs and has been mature for a decade, HIP, the close-to-metal choice for AMD GPUs, has only recently passed all the correctness tests for QMCPACK. We note that QMCPACK does not use any exotic or recent CUDA features. With the ROCm 4.5 release (October 2021), all QMCPACK core functionality tests pass for the first time on Radeon VII (gfx906) GPUs using our in-house test bed. While we were working on all the measurements for this report, ROCm 4.5 was not yet installed on Spock. For this reason, all the data collected in this report used ROCm 4.3. Although certain numerical results are incorrect by a small but scientifically significantly degree with ROCm 4.3, since the Monte Carlo acceptance ratio is within 1% of the correct value, we believe the amount of computation

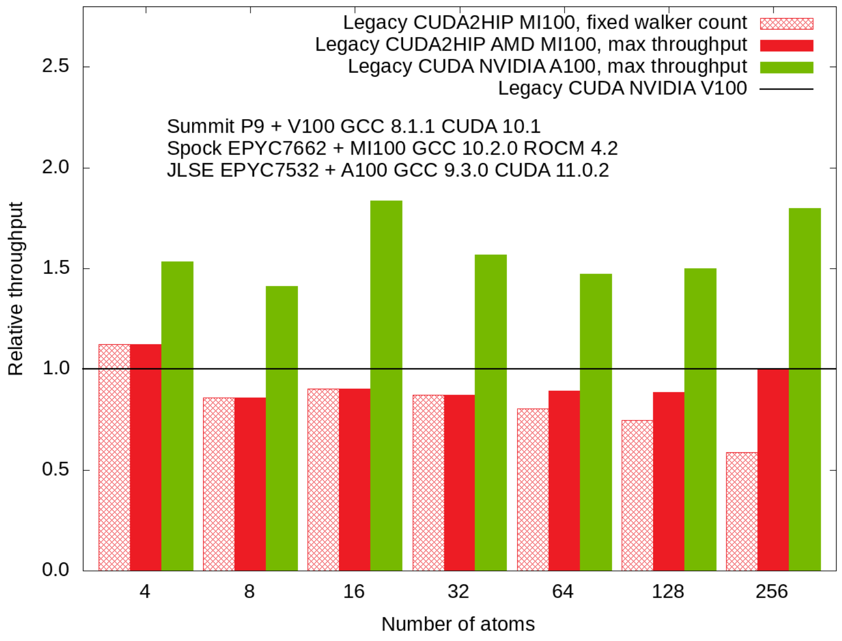


Figure 16: QMCPACK throughput vs problem size for HIP vs CUDA on V100, A100, and MI100.

is not significantly affected, the total numerical work is close to being correct, and therefore the measured performance is representative for AMD GPUs.

In Fig. 16 we compare the performance of QMCPACK legacy CUDA code running natively on NVIDIA V100 (baseline, 1.0), A100 and also on AMD MI100 GPUs with CUDA to HIP conversion (CUDA2HIP). When having the same workload, i.e. fixed walker count, as V100, MI100 (mesh red) runs more slowly for nearly all problem sizes. This indicates actual slower kernels on MI100 despite higher theoretical flops or higher memory bandwidth. The low kernel performance is likely caused by the well known limitations on Vega based architecture including CDNA 1.0 which only issues on a wavefront every 4 cycles and its 64-item wavefront is less efficient for low item counts. Moreover, the gap between MI100 and V100 actually becomes larger as the problem size grows. When running the 256 atom problem, both V100 and MI100 are underfed by workload due to the low walker counts. GPU runtime overhead cannot be hidden by computation and HIP runtime loses to CUDA runtime. When we add more walkers, using the increased GPU memory on the newer GPUs, MI100 (solid red) recovers more performance in large problem sizes as throughput increases with larger kernels.

By comparing CUDA and HIP with QMCPACK legacy CUDA code base, we are able to understand the current performance of AMD MI100 hardware and ROCm software. On one hand, they are behind what NVIDIA offers today. On the other hand, the performance difference is less than 50%. With this knowledge, we expect the performance portable version of QMCPACK to show a similar picture.

Let us first recap the current status and achievable throughput of performance portable QMCPACK on OLCF Summit as shown in Fig. 17. Summit node has two sockets of IBM P9 21 core CPUs and six NVIDIA V100 GPUs in total. All the runs place 6 MPI ranks per node and one GPU plus 7 cores are assigned to each MPI rank. All the throughput numbers in this report are per GPU. The performance portable QMCPACK uses OpenMP offload and CUDA libraries and thus labeled as “OpenMP+CUDA” in figures, but we will refer to it simply as the OpenMP offload version for convenience. With improved algorithms and implementation, 2.5× performance over legacy CUDA is achieved on the 512 atom problem. For smaller problem sizes, all the parallelization strategies beneficial for the 512 atom problem hurt performance, and a few memory saving algorithms for the 512 atom problem actually become a bottleneck. For these reasons, the compute tasks handled to GPUs are dispatched in walker batches for high performance for small to medium problem sizes. This is the strategy employed in the legacy CUDA code. Over the past years, we have seen performance

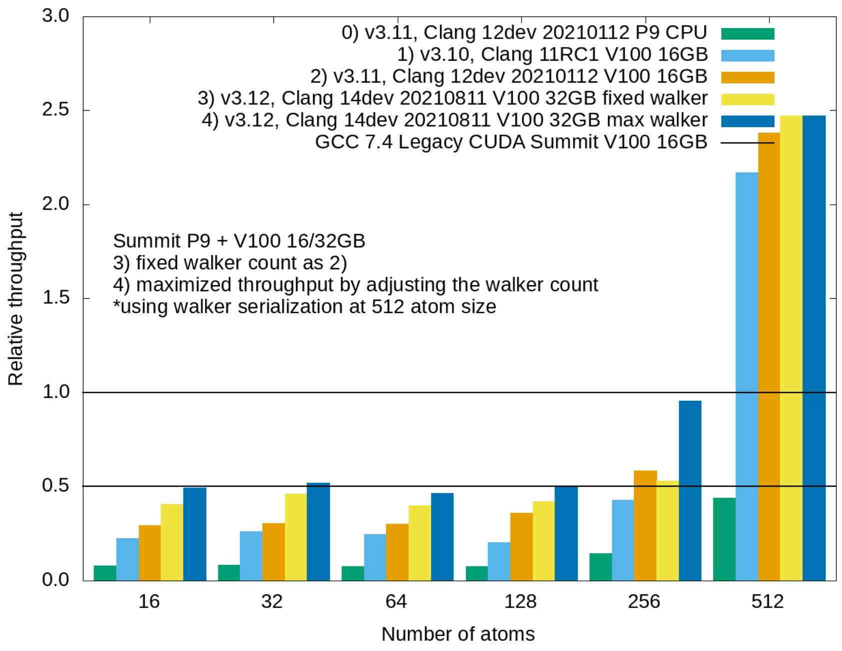
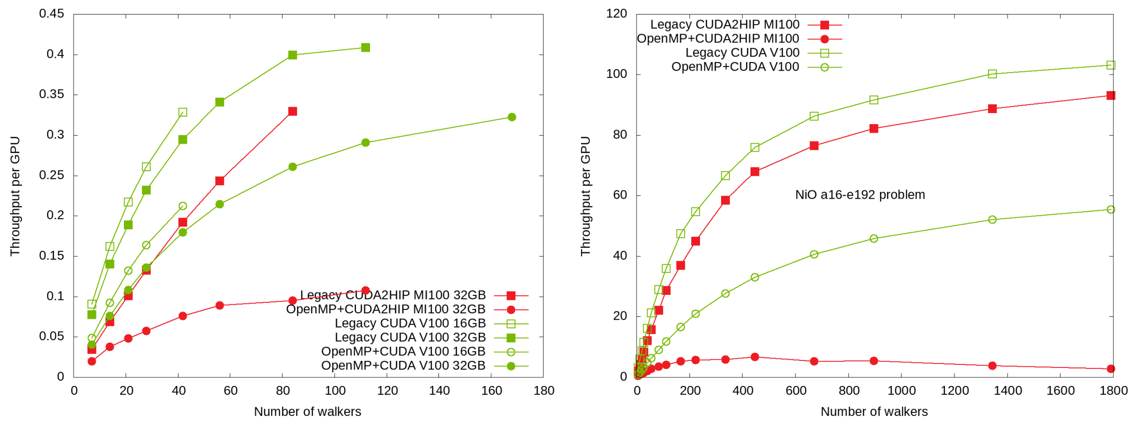


Figure 17: OpenMP offload performance on NVIDIA GPUs

improvements as we ported more computation to GPU and LLVM compiler OpenMP offload improves—the progression is shown via the cyan (v3.10, Clang 11), orange (v3.11, Clang 12), and yellow (v3.21, Clang 14) bars. At with the legacy CUDA and HIP benchmark, increasing the walker count using the increased memory of the newer GPUs (here 32 GB V100) results in increased performance. Note that there are only 7 walkers in total on per GPU for the 512 atom case. Currently, the throughput of OpenMP offload version is about 5× over CPU-only but 50% of legacy CUDA for all but the 512 atom problems benchmarked. We there expect its relative performance to legacy CUDA via HIP will be also 50% on MI100 for these problem sizes. Increasing the performance for smaller problems will require porting more features to the GPU, primarily to reduce data movement.

On Spock, each compute node has one AMD EPYC 7662 64 core CPU and 4 MI100 GPUs. The available OpenMP offload compilers are HPE Cray CCE and AMD ROCm LLVM compiler. Unfortunately, the Cray CCE compiler (v12 and v13) crashes in its OpenMP runtime when handling CPU tasks. The Cray compiler team is actively investigating this issue. As soon as the crashing issue is fixed, we will conduct a performance study. Although the ROCm 4.3 shipped LLVM compiler has OpenMP offload capability, it has incorrect GPU kernel generation and kernel execution hits a segmentation fault. The latest ROCm 4.5 release also does not resolve this issue. Since we reported this issue to AMD, the research compiler team provided us a fix via the AOMP 14.0 compiler. Since AOMP 14.0 was unreleased when all the measurements were conducted, we built it from source code using the ROCm 4.3 tool chain, thereby making performance portable QMCPACK with OpenMP + HIP working on Spock. Due to the fact that the MPI installation on Spock is tightly coupled to the ROCm tool chain, it doesn't work with our hybrid AOMP + ROCm mixed version stack. Therefore all the following runs do not use MPI, and they only run on 16 cores with one MI100 GPU. On Spock, the EPYC processor is configured in NUMA mode and 16 cores map to a single NUMA. So the GPU timings should be the same as running 4 MPI tasks on the whole node since MPI ranks are used for weak scaling only.

Figure 18a shows the throughput of 256 atom runs with different walker counts. The performance of both legacy CUDA and the performance portable version on NVIDIA and AMD improves as the walker count increases. When both are using the legacy CUDA code base, the MI100 is about 80% of a V100. On the V100, the offload version is also about 80% of legacy CUDA. However, on the MI100 the offload code performance is only 30% of the HIP version of the legacy CUDA code. This is lower than expected. Figure 18b shows the throughput of the 16 atom problem. The OpenMP offload performance is only 1/20 of legacy CUDA code via HIP on the MI100. Figure 19 shows the throughput for all the problem sizes. The OpenMP offload



(a) **(b)**

Figure 18: Performance for (a, left) 256 and (b, right) 16 atom NiO supercells, comparing of native vs OpenMP offload implementations.

version on the MI100 is significantly worse than running on the V100.

We have analyzed and followed the performance of ROCm and AOMP over the last few years based on our code timers/traces and by looking at the AMD source code. Multiple issues have been raised. For example, rocprof would crash with some OpenMP offload code, precluding profiling with code generated by AMD's compiler. This is now fixed in unreleased development versions, but we were not able to make it work with our hybrid AOMP-ROCm 4.3 compilation on Spock.

Several significant performance issues remain, including:

- Oct 8th 2020. Runtime overhead on H2D and D2H transfers https://github.com/ROCm-Developer-Tools/aomp/issues/160
- Oct 8th 2020. Reduce synchronization needed https://github.com/ROCm-Developer-Tools/aomp/issues/161
- Nov 4th 2021. The additional thread for GPU https://github.com/ROCm-Developer-Tools/aomp/issues/257

Note that two of these issues have been open for over 1 year and the equivalent problems do not exist for NVIDIA targets with LLVM. We have clearly shown that maturation of the compilers targeting AMD is required to reach acceptable performance as compared to competitor products.

4. ENERGY APPLICATIONS

End State: Deliver a broad array of science-based computational applications able to provide—through the effective exploitation of exascale HPC technologies—breakthrough modeling and simulation solutions that yield high-confidence insights into a set of critical problems and challenges that positively impact the nation's energy security.

The energy applications (EA) L3 area (Table 27) focuses on modeling and simulating existing and future technologies for the efficient and responsible production of energy to meet the growing needs of the United States. The applications in this WBS L3 generally require the detailed modeling of complex facilities and multiple coupled physical processes. Their goal is to help overcome obstacles to efficiently and safely deliver energy.

These applications are highly complex and involve modeling intricate geometric details and including a broad range of physical phenomena. One key additional requirement for EA is the broader community adoption of the computational models and methods developed in the project or, in some cases, the virtual datasets or physical insights that result from simulations carried out in the ECP. Additionally, applications are expected to influence—directly or indirectly—design choices for exascale hardware and software.

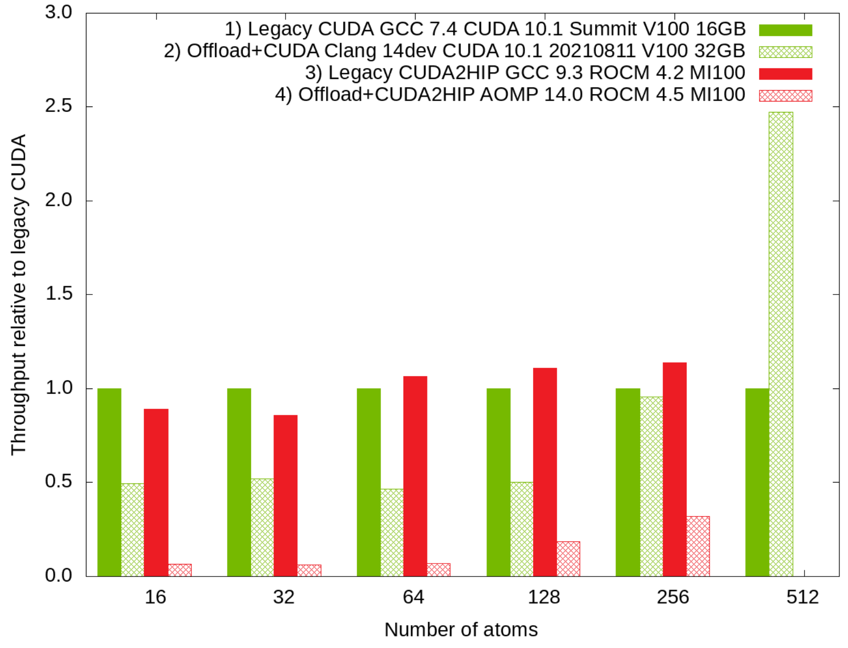


Figure 19: Comparison of OpenMP offload performance on AMD vs NVIDIA GPUs

Table 27: Summary of supported EA L4 projects.

WBS number	Short name	Project short description	KPP-X
2.2.2.01	ExaWind	Predictive wind plant flow modeling	KPP-2
2.2.2.02	Combustion-Pele	Combustion engine and gas turbine design	KPP-2
2.2.2.03	ExaSMR	Coupled MC neutronics and fluid flow simulation of small modular reactors	KPP-1
2.2.2.04	MFIX-Exa	Multiphase flow reactor design	KPP-2
2.2.2.05	WDMApp	High-fidelity whole device modeling of magnetically confined plasmas	KPP-1
2.2.2.06	WarpX	Plasma wakefield accelerator design	KPP-1

4.1 ExaWind

4.1.1 KPP Verification Plan

Minimum and stretch challenge-problem requirements

The minimum challenge-problem simulation will contain at least four megawatt-scale turbines (e.g., National Renewable Energy Laboratory (NREL) 5 MW reference turbines) organized in a 2×2 array and residing in a $3 \times 3\,\mathrm{km}^2$ domain with a height of at least 1 km. A hybrid Reynolds-averaged Navier-Stokes/large eddy simulation (RANS/LES) model will be employed for which a Reynolds-averaged Navier-Stokes (RANS) model will be used near turbine-blade surfaces, and an large eddy simulation (LES) model will be used in the wake region. The simulation will have a mean wind speed at the turbines' rated speed (e.g., $11.4\,\mathrm{m\,s^{-1}}$ for the NREL 5 MW reference turbine). The model will require at least 20 billion gridpoints and 100 billion DOF to resolve the system, and near-blade grid spacing will be such that the viscous sublayer within the RANS region is resolved. A successful simulation will require an optimized solver stack that minimizes the wall-clock-time per time step. A scientifically meaningful simulation duration will be for at least one domain transit time (about 370 s for the $3 \times 3\,\mathrm{km}^2$ domain at $11.4\,\mathrm{m\,s^{-1}}$). The team will demonstrate that such a simulation is feasible within 4 weeks of system time. The stretch-goal challenge-problem simulation will contain at least forty turbines in an area of at least $50\,\mathrm{km}^2$.

Verification simulations

At the center of the ExaWind project is the open-source ExaWind software stack, which is composed of three, coupled, physics-based solvers: Nalu-Wind, AMR-Wind, and OpenFAST. Nalu-Wind is an unstructured-grid incompressible-flow computational fluid dynamics (CFD) code that is built on Trilinos and is used to resolve the fluid dynamics around wind turbine blades. Nalu-Wind turbine meshes are embedded in a background AMR-Wind mesh and are coupled through the overset method handled by the TIOGA library. AMR-Wind is a structured-grid, incompressible flow CFD code built on the AMReX (§ 7.3) library with adaptive-mesh-refinement capabilities and is well suited to resolve wind turbine wake propagation. OpenFAST is a whole-turbine simulation code, which includes models for blades, drivetrain, tower, control system, etc.

Our challenge-problem simulation will be accomplished through our hybrid solver approach, wherein AMR-Wind and Nalu-Wind are solved separately but concurrently, and coupled via overset meshes in an additive-Schwarz manner [14]. There will be a Nalu-Wind instance for each turbine and each instance will have about 10 M gridpoints. Nalu-Wind can strong scale well down to 250 k gridpoints per GPU [15]. Below that threshold, CPUs are better performing. In the challenge-problem simulation, there will be roughly 20 B AMR-Wind gridpoints and 40 M Nalu-Wind gridpoints (four turbines at 10 M gridpoints each). If AMR-Wind and Nalu-Wind were run on GPUs at their strong-scaling limits, for example, based on preliminary Summit calculations (see Table 28), they would require roughly 28% and 0.5% of Frontier, respectively, if we assume that Frontier will have 9000 nodes with four GPUs per node. However, the simulation time per timestep is likely to be different between Nalu-Wind and AMR-Wind, which would have GPUs remaining idle while the slower code catches up. In the scenario that Nalu-Wind is slower than AMR-Wind, we will either add more refinement to the AMR-Wind model to capture more physics or we will distribute the 20 M AMR-Wind gridpoints amongst fewer GPUs, keeping the time-per-timestep approximately equivalent between the two CFD codes. In an alternative scenario, we may run Nalu-Wind on the CPUs if shorter time per timestep can be achieved there. The ExaWind team, however, hopes to push well beyond the minimum requirements and to simulate a large wind farm with 40 or more wind turbines using over half of Frontier. The ExaWind team will perform strong- and weak-scaling studies on Frontier in order to inform the appropriate distribution of model gridpoints across CPUs and GPUs in order to achieve the shortest time to solution of the full challenge problem. There will be an OpenFAST instance for each turbine in the wind farm, and each OpenFAST instance will be run on a single CPU-core. The team will document GPU and CPU based performance and compare that performance against baseline GPU and CPU preliminary calculations on Summit.

Simulation artifacts

The ExaWind application consists of an application driver (ExaWind-Driver) which is essentially run similar to a multi-program multi-data (MPMD) mode. The three applications running in tandem, Nalu-Wind, AMR-Wind, and OpenFAST each produce their own log files (for each instance) and set of output data

Table 28: Approximate Summit-based strong-scaling limits for AMR-Wind and Nalu-Wind.

Code	CPU strong-scaling limit	GPU strong-scaling limit
AMR-Wind	1.5×10^4 gridpoints/core	2.0×10^6 gridpoints/GPU
Nalu-Wind	2.0×10^4 gridpoints/core	2.5×10^5 gridpoints/GPU

specified separately in each application. Each application is also able to generate visualizations in-situ. However, Nalu-Wind output should be within a manageable size to allow for post processing. AMR-Wind mesh output over the full domain would require ZFP lossy compression capability to allow for full post processing of the entire domain. Verification of successful demonstration should entail inspection of the outputs similar to the following:

1. `job_name.o1234` : Log file output from the job and the top-level application driver capturing `stdout` and `stderr`, provides evidence of successful timestepping for N timesteps.
2. `nalu-wind.log` : Detailed log of timestepping within each Nalu-Wind instance, capturing convergence information of the linear solvers, and number of elements.
3. `amr-wind.log` : Detailed log of timestepping within AMR-Wind, capturing convergence information of the linear solvers, and number of elements.
4. `5MW_Land_BD_DLL_WTurb.log` : Detailed log of timestepping within each OpenFAST instance, with one instance per wind turbine.
5. `output/nalu-wind-mesh-nalu_instance_num.exo.rank_num` : Exodus mesh files in a directory, providing the initial condition for each Nalu-Wind instance in the simulation, as well as mesh output for the Nalu-Wind instances' final conditions.
6. If visualization of the full domain is not feasible in AMR-Wind, in-situ visualization capabilities using Ascent can be utilized by default:
   (a) `amr-wind-image-0.png` : Slices of AMR-Wind domain at initial condition,
   (b) `amr-wind-image-10.png` : Slices of AMR-Wind domain at final condition.
7. Provided ZFP-type lossy compression is available in AMReX at the time of demonstration, it may be possible to post-process and view the entire domain of the simulation in Paraview. Paraview is able to load the exodus mesh file output from Nalu-Wind and overlay it with the correct displacement within the AMR-Wind mesh plot files. Therefore AMR-Wind could output full domain plot files:
   (a) `plt00000` : Directory providing plot mesh data for the AMR-Wind domain initial condition for velocity, vorticity, and temperature,
   (b) `plt00010` : Directory providing plot mesh data for the AMR-Wind domain final condition for velocity, vorticity, and temperature.

Confirmation of minimum requirements

Post-processing of outputs (shown below) will provide confirmation satisfaction of minimum requirements.

1. `job_name.o1234`, `nalu-wind.log`, `amr-wind.log`, `5MW_Land_BD_DLL_WTurb.log` : All inspected for sanity, successful completion of timesteps, and reporting of time per timestep.
2. `amr-wind-image-0.png` , `amr-wind-image-10.png` : Ascent images for AMR-Wind and selected quantities viewed for expected transition from initial condition to final condition.
3. `output/nalu-wind-mesh-nalu_instance_num.exo.rank_num` : Loaded into Paraview for visualization of pertinent physical quantities and expected transition from initial condition to final condition.

Table 29: Status of ExaWind physics solvers on Arcticus.

Component	Device	Status
ExaWind-Driver	GPU	N/A
ExaWind-Driver	CPU	Build not yet attempted, but only requires C++17 compiler
AMR-Wind	GPU	Builds and runs test suite, although not fully tested with optional libraries such as NetCDF
AMR-Wind	CPU	Builds and runs test suite, although not fully tested with optional libraries such as NetCDF
Nalu-Wind	GPU	Waiting on Kokkos, Trilinos/STK, and virtual function support on device
Nalu-Wind	CPU	Known to build with a previous Intel OneAPI release
OpenFAST	GPU	N/A
OpenFAST	CPU	Known to build with a previous Intel OneAPI release

4. If `plt00000` , `plt000010` : Also loaded into Paraview along with Nalu-Wind files to visualize and inspect transition of physical quantities from initial to final conditions. VisIt is also an option for visualizing AMR-Wind plot files, without the ability to overlay Nalu-Wind meshes.

Additionally, if the ExaWind team has access to Frontier node hours for science-based runs (e.g., via INCITE), the team will provide, e.g., results isocontour snapshots, turbine power results, and flow movies.

4.1.2 Early Hardware Status

On current Nvidia-based GPU machines, the ExaWind hybrid solver, known as the ExaWind-Driver, has been verified to run successfully on the Summit and Eagle machines in a configuration where both AMR-Wind and Nalu-Wind are utilizing the GPUs for the simple flow over a sphere test case. The configuration where AMR-Wind is run on the GPU and Nalu-Wind is run on the CPU has also been run successfully on Summit and Eagle. This provides the ExaWind team a pathway to explore coupled simulations on the early hardware in parallel to the work of enabling Nalu-Wind on the new GPU architectures.[1] With this as the status of ExaWind on Summit and Eagle as our baselines for GPU-enabled machines, in the following, we describe status of the ExaWind stack on the early hardware systems Arcticus and Spock.

Arcticus/Intel

In general, two issues have slowed progress on the early Intel JLSE machines at Argonne: (1) Intel made changes to the terms of use for the JLSE machines which many laboratories were reluctant to sign for some time. (2) The migration of the JLSE machines to a new accounting system. Both of these changes hindered the ExaWind development team's access to the early machines, as well as for the teams of the upstream libraries ExaWind relies upon. Status details for Arcticus are listed in Table 29.

Spock/AMD

Substantial progress has been made on Spock with ExaWind solvers, with status summarized in Table 30 for the primary physics solvers.

A comparison for AMR-Wind between a single MI100 and a V100 is shown in Fig. 20. An AMReX from October 2021 was used with ROCm 4.3.0 on Spock for the MI100, and CUDA 11.2.2 with GCC 8.4.0 on Eagle for the V100. Performance is comparable to the V100. When comparing profiling results between GPUs, no particular routines are found to be significantly slower on the MI100. Also, newer versions of ROCm are available on Tulip which may have improved performance as well.

[1]Along with this flexibility, the ExaWind-Driver will likely benefit from, or possibly even require advanced MPI rank mapping functionality using something such as Explicit Resource Files (ERF) [16], as is available on Summit.

Table 30: Status of ExaWind physics solvers on Spock.

Component	Device	Status
ExaWind Driver	GPU	N/A
ExaWind Driver	CPU	Build not yet attempted, but only requires C++17 compiler
AMR-Wind	GPU	Builds and runs test suite, though not tested with optional libraries such as NetCDF
AMR-Wind	CPU	Builds and runs test suite, though not tested with optional libraries such as NetCDF
Nalu-Wind	GPU	Waiting on STK functionality in Trilinos and virtual function support on device, where STK currently has some unit tests functioning
Nalu-Wind	CPU	Currently working to resolve build issues encountered with Trilinos, although Trilinos is known to build and test successfully on Spock
OpenFAST	GPU	N/A
OpenFAST	CPU	Build not yet attempted

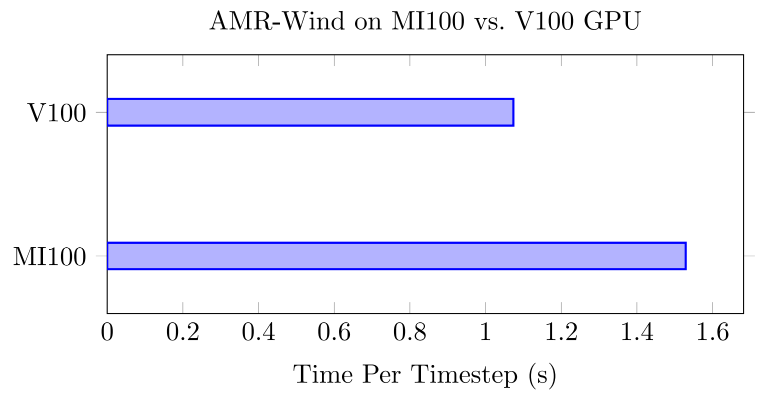


Figure 20: AMR-Wind atmospheric-boundary-layer regression test (`abl_godunov`) with 192^3 domain on a single MI100 vs. a single V100.

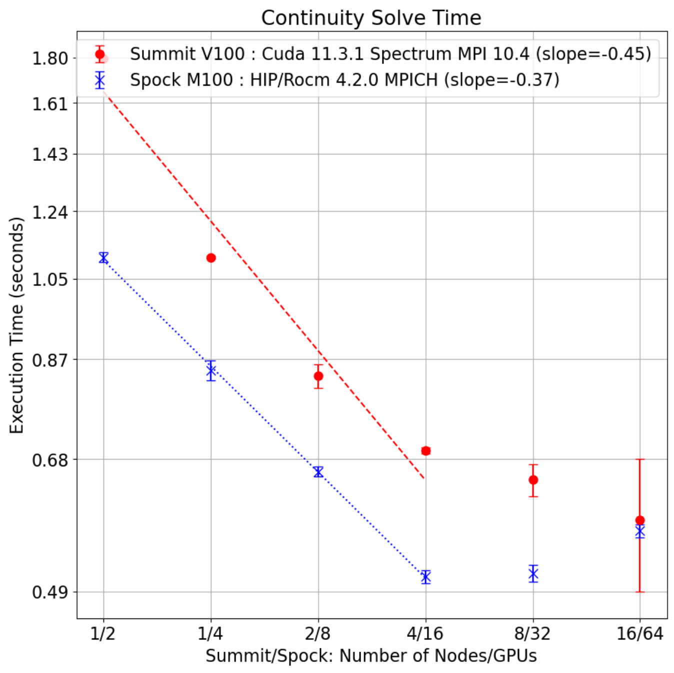


Figure 21: ExaWind Summit and Spock: pressure-Poisson hypre solve costs for the matrix associated with the first time step in the NREL-5-MW-turbine simulation. We used four V100 GPUs per node on Summit and four M100 GPUs per node on Spock. Performance for a hypre branch containing optimizations, that are not yet merged to the master branch, are given.

Due to the lack of virtual function support on AMD GPUs, we cannot run the full Nalu-Wind or ExaWind Driver application on Spock. However, we can measure the performance of the critical solver libraries, hypre, AMReX, and Trilinos:Solvers outside of these applications on linear systems directly related to the challenge problem. The current status, performance, issues and risks are summarized in Table 31. We have found that the performance of these particular solvers measured outside of the full application are a strong predictor of the performance on most GPU hardware, when fully embedded.

The strong scaling performance of the hypre implementation of the most time-consuming linear system solve is shown in Fig. 21. The performance improvement of the M100 over the V100 is encouraging. We predict that the Frontier system will demonstrate even better performance for this solve thus enabling a much faster time per time step when running in the full Nalu-Wind/ExaWind Driver application.

A critical algorithm component of the AMR-Wind background solver are the AMReX MLMG solvers. Figure 22 shows the performance of the AMReX Nodal Poisson regression test. This test leverages the MLMG software stack and thus its performance is critical to the AMR-Wind/ExaWind Driver application performance. In this case, the M100 performance is slower than the corresponding V100 performance for Summit. This suggests that optimization of MLMG algorithms for AMD architectures is still necessary in order to take full advantage of AMD GPU potential.

4.2 Combustion-Pele

4.2.1 KPP Verification Plan

The Pele project's KPP-2 challenge problem is derived from the road map toward the motivating exascale era problem. Specifically, the challenge problem demonstrates the ability to simulate the interaction of two fuels with varying reactivity under a multi-pulse injection strategy into engine-relevant geometry. It is a baseline for a series of simulations that will enable the impacts of multi-physics effects to be isolated, such as spray evaporation on mixture fraction and temperature, alternative fuels, soot emissions and the design of strategies to control combustion phasing and subsequent combustion rates. The problem will be tractable

Table 31: Status of critical ExaWind solver libraries on Spock (hypre, AMReX, and Trilinos: Solvers).

Component	Comments
hypre	Status: · Package successfully builds with AMD GPU support and without required managed memory · Performance critical algorithms including matrix assembly, preconditioner setup, and solve are under active development Performance results for critical problems: · Matrices for the 23M-gridpoint Nalu-Wind blade-resolved NREL-5-MW-turbine mesh were used for strong scaling studies out to 16 Spock nodes · 30-40% improvement over Summit for pressure-Poisson and momentum equations with degraded or equivalent scaling; see Fig. 21 for pressure-Poisson performance results Issues and risks: · Degraded solver scaling and performance volatility beyond 16 M100 GPUs requires further investigation
AMReX	Status: · Package successfully builds with AMD GPU support · Currently investigating optimizations to GPU-MPI messaging performance Performance results for critical problems: · Strong scaling of key linear solver algorithms shows degraded performance compared to Summit; see Fig. 22 Issues and risks: · Degraded performance of key linear solvers on the M100 (Spock) compared to V100 (Summit) requires further investigation
Trilinos: Solvers	Status: · Working on performance improvements across the linear-system solver stack, this include kernel launch heuristic improvements, further integration of vendor TPLs (rocBLAS, rocSPARSE). Currently the focus is on improved strong/weak scalability in the multigrid solver and on kernels performance improvement (Matrix-Vector product and Matrix-Matrix product). · The Trilinos linear-system solver stack has been fully tested on multiple compute nodes and is ready for use within Nalu-Wind.

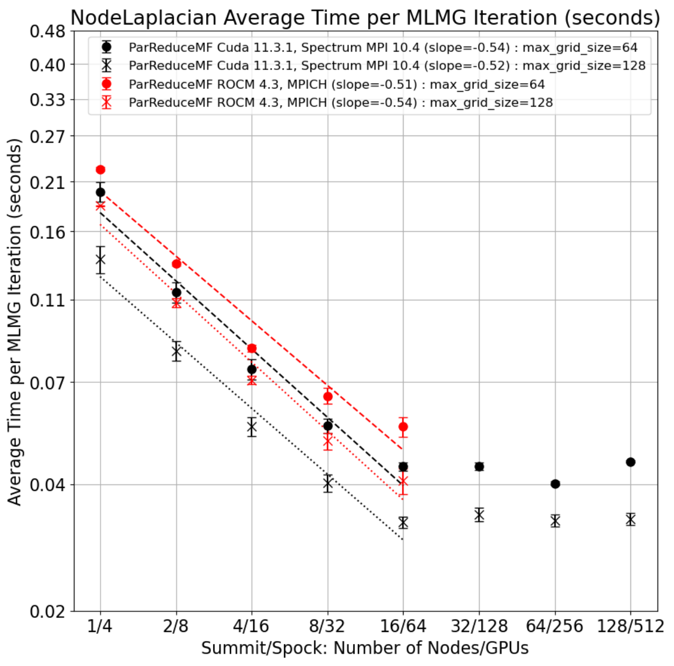


Figure 22: ExaWind Summit and Spock strong-scaling performance of AMReX Nodal Poisson-system solve on domain of size 512^3 for $max\_grid\_size = 64, 128$ and the AMReX checkpoint 69b69c2b281c. We used four V100 GPUs per node on Summit and four M100 GPUs per node on Spock.

under a realistic allocation by using the full capabilities of an exascale machine; within the projected 50–100× increase in productive capability of Frontier beyond the capabilities of Titan, the simulation will execute in approximately 2–4 weeks of wall time.

Dependencies 400 TB restart file (created on all of Summit), SUNDIALS, hypre, AMReX (§ 7.3), MAGMA.

Inputs bash/batch script files, pre-computed inert velocity fluctuation data (1 TB) to seed jet turbulence, 400 TB restart file at the beginning of a time window identified from a coarse-mesh preliminary simulation after the main injection has undergone low-temperature auto-ignition and is transitioning to high-temperature ignition. Run will require access to approximately three-quarters of Frontier CPU/GPU resources.

Verification of KPP-2 Threshold

- The code utilized exascale resources:
  Collect stdout output for a successful time step, and confirm successful output of several field variables (e.g. temperature, species mass fractions, velocity magnitude, etc.) as advanced over the 10–20 time steps. As a sanity check species mass fractions and temperature ranges should be consistent with lower-resolution reference simulations and lower-dimensional reference simulations (e.g., homogeneous reactor).

- The executed problem met challenge problem minimum criteria:
  Successfully generate in-situ statistics that confirm reasonable measure of the progress of auto-ignition and/or integrated chemical heat release rate and/or fuel consumption rate over the run interval. The preliminary measure of ignition is defined as the mass fractions of formaldehyde and ketohydroperoxide exceeding 1.75×10^{-2} and 4.5×10^{-2}, respectively, for low-temperature ignition, and temperature and carbon dioxide mass fraction exceeding 2200 K and 0.12, respectively, for high-temperature ignition. These numbers are based on zero-dimensional auto-ignition calculations with n-dodecane injection into

Table 32: Combustion-Pele challenge problem details.

Functional requirement	Minimum criteria
Physical phenomena and associated models	Multiscale CFD with reacting flows in low-Mach and/or compressible formulations with DNS and LES turbulence resolution with multispecies chemistry.
Numerical approach and associated models	Time-explicit and deferred correction strategies in a compressible and projection-based low-Mach formulation, respectively. Finite volume spatial discretizations on block-structured AMR grids with embedded boundaries. Hybrid DNS/LES to enable fully resolved (i.e., DNS) treatment where turbulence chemistry interaction occurs and modeled (i.e., LES) treatment to reduce resolution requirements in non-reacting portions of the flow.
Simulation details	Gas-phase simulation of four multi-pulse jets interacting in a $\frac{1}{4}$ scale geometry (approximately 2.5 cm diameter) derived from a production engine piston bowl with a flat chamber ceiling, shaped piston head and radially centered fuel injector. One or two pulses of a high-reactivity fuel into a low-reactivity premixture that capture cross-mixing and reactions between fuels by using $\sim$ 30–35 species or more; 1.0 ms physical simulation time or more; up to four levels of hierarchical mesh refinement; finest grid 1.25 μm; and a realistic chamber environment of $>$ 50 bar.
Demonstration calculation	Restart from a checkpoint that provides a realistic development of the pulsed jet into geometry obtained by running the case with restricted resolution where additional refinement is added at the time of restart. Subsequently, run 10–20 time steps to compute a realistic grind time that can be used to estimate the cost of the full-time horizon.

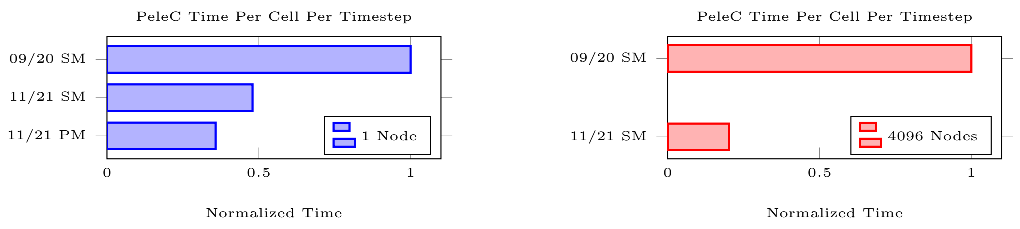


Figure 23: PeleC time per cell per timestep from September 2020 code using `drm19` 21 species mechanism on Summit compared to November 2021 code using `dodecane_lu` 53 species mechanism on Summit and Perlmutter. The 53 species case is linearly scaled to approximate a 21 species run time. Indicates a 2× speedup on a single Summit node and 5× speedup on 4096 Summit nodes over this time period.

a methane-oxidizer environment at 60–110 atm (it should be noted that turbulent mixing can alter peak values by as much as 40% compared to zero-dimensional auto-ignition calculations). We will add an additional metric to PeleC and PeleLM, based on multiple levels of refinement available during the run which will estimate key discretization errors.

4.2.2 *Early Hardware Status*

PeleC

Before discussing the status of the Pele suite on early-access hardware, we first discuss a few improvements to PeleC since the review in 2020. Most notably, AMReX has significantly improved key impediments to weak scaling in both Pele codes. PeleC was also a part of a GPU hackathon in July 2021 which targeted improvements in the chemistry calculations. In trying to quantify these improvements, we face a number of minor complexities. First, after significant interactions with the chemistry development team during FY21, we have identified a 53-species "skeletal" mechanism, `dodecane_lu`, that is appropriate for use in the anticipated challenge problem fueling scenario. All of our benchmarks cases have been updated to run with this larger/stiffer 53 species mechanism, rather than the 21-species placeholder model that was used for benchmarking in prior years. In addition, AMReX has modified some of the detailed processes to generate grid-lists from cells tagged for mesh refinement, resulting in grid layouts that are inconsistent with those of prior tests. Finally, where SUNDIAL's explicit time integrators, ARKStep/ERKStep, were used in prior years, optimizations during the GPU hackathon have made SUNDIAL's implicit integrator, CVODE, the current method of choice for maximum performance on the GPU (for both PeleC and PeleLM). All of these factors complicate the generation of quantitative metrics to track our improvements to overall code efficiency. Figure 23 displays the normalized time-per-cell-per-timestep–here, the 53-species case data was linearly scaled to approximate timings from an equivalent 21-species run. The data suggests a 2× speedup on a single node of Summit and a 2.8× speedup when comparing a single Summit node with code taken from September 2020 to a single node of Perlmutter with code from November 2021. The majority of the observed speedup was due to optimizations performed during in the GPU hackathon. On 4096 Summit nodes, a speedup of 5× is observed, due primarily to these chemistry optimizations and to improvements in the weak-scaling behavior of AMReX data structures that underlie the Pele codes.

Strong and weak scaling for PeleC is updated in Figure 24 using the 53-species `dodecane_lu`, as well as using CVODE as the preferred chemistry integrator. Since the overall runtime of routines in PeleC, as well as our total runtime, have decreased over time, we see our scaling suffers since past iterations of the benchmark. No longer are the chemistry routines the most significant contributors to runtime, even with the larger 53-species mechanism. MPI communication through AMReX has now become a significant bottleneck in the code as illustrated in Figure 25. Note that Perlmutter was in early stages of acceptance at the time of running the scaling study there, so its performance is likely not indicative of its future capability. Also of note in the weak scaling plot is that we are running with the maximum amount of cells that can fit within the memory of the Summit V100 GPUs, which is around 700 k per GPU for the `dodecane_lu` case. To achieve a smaller time per timestep, there is some room for reducing the number of cells per GPU without completely giving up parallel efficiency. We project that on the entire Summit machine, we can currently advance approximately 20 B cells for one time step in approximately 10 seconds.

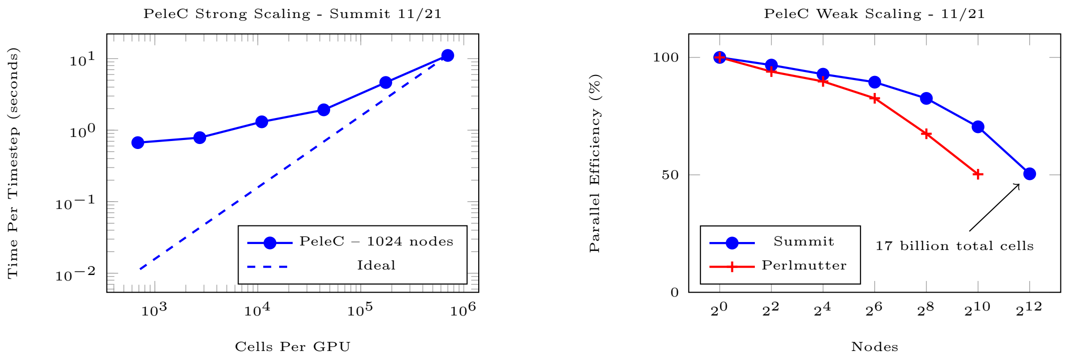


Figure 24: PeleC strong (left) and weak (right) scaling of PMF case with `dodecane_lu` chemistry on 1024 Summit nodes. Varying number of cells with one level of AMR.

In regards to early hardware status. All of PeleC's functionality is now available on the AMD ROCm platform. Using a single MI100 on Tulip and the ROCm 4.5.0 release, Figure 26 shows a comparison to results from a V100 on Eagle using CUDA release 11.2.2. PeleC is now within 20% of the performance between the MI100 and V100 when using an early version of SUNDIALS 6.0. PeleC does not yet run on Spock due to a requirement of ROCm 4.5.0 which resolves certain compiler bugs encountered by PeleC. Pele team members did not have access to Crusher at the time of this report.

Most of PeleC's functionality is available for Intel's OneAPI and DPC++ framework. PeleC can run demonstration problems on Arcticus and Iris, however, performance data has not yet been collected. Embedded boundaries rely on Thrust on the GPU, and we have not yet begun the effort of porting the Thrust calls to OneDPL for the Intel platform.

PeleLM

Following PeleC example, PeleLM performances on pre-ExaScale systems are first revisited using the 53-species `dodecane_lu` mechanism selected for the challenge problem and the most recently available option to perform the implicit stiff chemistry integration. In particular, the CVODE integration is now performed using the batched block-dense direct solve provided by the MAGMA library to invert the Newton systems within CVDOE's internal subcycling strategy. This new configuration enables a 3x speedup compared to our

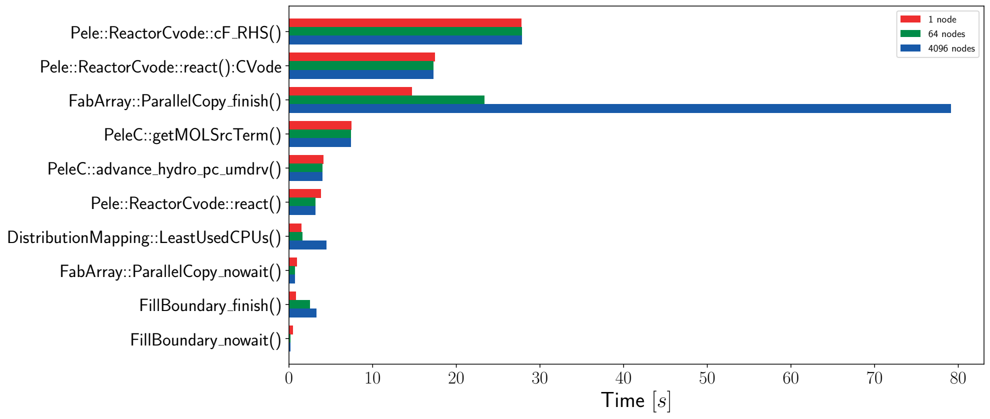


Figure 25: Profile breakdown of PeleC routines as weak scaling is increased.

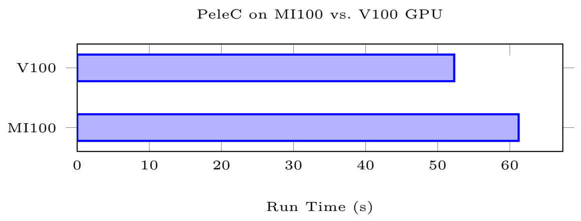


Figure 26: PeleC PMF `dodecane_lu` case for 10 timesteps on an MI100 GPU on Tulip with ROCm 4.5.0 vs. a V100 GPU on Eagle with CUDA 11.2.2.

previous best option (matrix-free GMRES solve) on the "PMF" case used as our primary benchmark case. Also note that, compared to a January 2021 PeleLM version, an additional 1.9× speedup of PeleLM single node performance was obtained by grouping together the species diffusion linear solves and reducing the number of `FillPatch` operations (used to fill grow cells prior to stencil operations).

Figure 27 shows PeleLM weak scaling performance on Perlmutter and Summit, and strong scaling on Perlmutter. The parallel efficiency remains above 50% up to 4096 Summit nodes and improvements of the parallel efficiency are obtained on Perlmutter with a 85% efficiency at 1024 nodes versus 73% on Summit. As in the case of PeleC, the reduction of PeleLM base runtime tends to more critically highlight the overhead of AMReX's MPI communications.

The plots also highlight the limited ability of PeleLM to strong scale effectively. In particular, while the chemistry integration component of the time steps scales only slightly below optimum (due to load imbalances), the linear solvers key to PeleLM implicit diffusion strategy exhibit almost no runtime improvements as number of nodes increases.

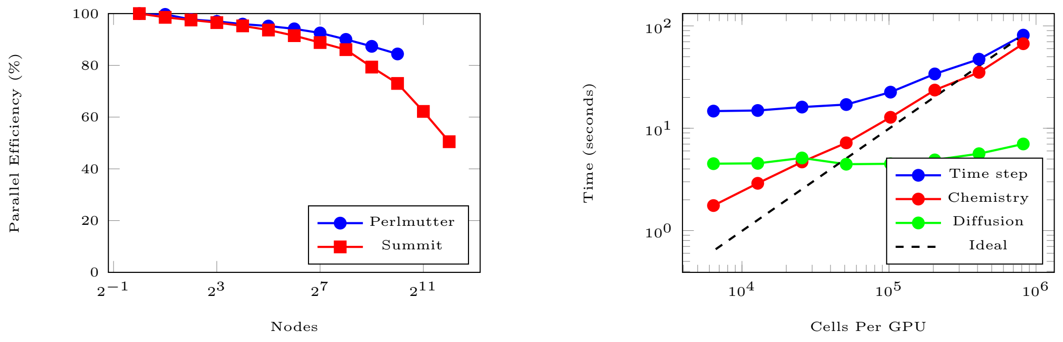


Figure 27: Weak (left) and strong (right) scaling of PMF case with `dodecane_lu` chemistry, using CVODE+MAGMA chemical integrator on Summit and Perlmutter. A constant 750k cells per GPU is employed.

To enable PeleLM on early access hardware, PeleLM memory handling was revised to remove all of the managed memory and most of PeleLM functionalities are now available on AMD ROCm platform (using ROCm 4.5 for the same reasons as PeleC), with similar performances comparing M100 versus V100 as reported for PeleC using an early release of SUNDIALS 6.0. Ongoing work focuses on enabling/testing the CVODE+MAGMA implementation on Tulip as well as using (and improving) the hypre library to solve for PeleLM's linear systems (with AMD ROCm).

4.3 ExaSMR

4.3.1 KPP Verification Plan

The ExaSMR challenge problem is the simulation of a representative NuScale small modular reactor (SMR) core by coupling continuous-energy MC neutronics with CFD. Features of the problem include the following:

- representative model of the complete in-vessel coolant loop,

Table 33: ExaSMR challenge problem details

Functional requirement	Minimum criteria
Physical phenomena and associated models	Eigenvalue form of the linear Boltzmann transport equation with quasi-static nuclide neutronics coupled to hybrid RANS/LES (or equivalent accuracy incompressible CFD with Boussinesq approximation, or low-Mach incompressible CFD with nonzero thermal divergence.
Numerical approach and associated models	The neutronics solver is MC particle transport. The CFD solver is a spectral finite element on unstructured grids. Physics are coupled by using a quasi-static approximation.
Simulation details	The neutronics model has a minimum of 200,000 tally bins and 10 billion particle histories per eigenvalue iteration. The CFD model has a minimum of 200 million elements and 70 billion DOF .
Demonstration calculation requirements	Run 30 eigenvalue cycles (10 inactive and 20 active) to estimate the MC particle tracking rate and 1000 time steps toward steady-state convergence in the CFD solve. Only one nonlinear (Picard) iteration is required. To facilitate comparison with the baseline measurement, the neutronics portion of this calculation requires six macroscopic reaction rates and one radial region per pin.

- hybrid RANS/LES turbulence model or RANS plus an LES-informed momentum source for treatment of mixing vanes, and
- pin-resolved spatial fission power and reaction rate tallies.

Details on the challenge problem specification are given in Table 33.

The simulation objective is to calculate reactor start-up conditions that demonstrate the initiation of natural coolant flow circulation through the reactor core and primary heat exchanger. The driver application, ENRICO, performs inline coupling of the Nek5000 CFD module with MC through a common API that supports two MC modules: Shift, which targets the Frontier architecture at Oak Ridge National Laboratory (ORNL), and OpenMC, which targets the Aurora system at Argonne National Laboratory (ANL).

Minimum neutronics requirements for the coupled simulation are:

- full-core representative SMR model containing 37 assemblies with 17×17 pins per assembly and 264 fuel pins per assembly item,
- depleted fuel compositions containing $O(150)$ nuclides per material,
- 10^{10} neutrons per eigenvalue iteration,
- pin-resolved reaction rate with a single radial tally region and 20 axial levels, and
- six macroscopic nuclide-independent reaction rate tallies.

Minimum CFD requirements for the coupled simulation are:

- assembly bundle mesh models with momentum sources from a resolved CFD calculation on a representative spacer grid and
- full-core mesh 200×10^6 elements and 70×10^9 degrees of freedom (DOF) .

Demonstration Calculation

Facility Requirements

Table 34: Summary of coupled physics input files.

Component	Filename	Description
MC	xyz.inp.xml	Driver input
	xyz.rtk.xml	Geometry input
	xyz_compositions.h5	Material definitions
CFD	xyz.udf	User-defined C++ functions
	xyz.oudf	User-defined OCCA kernels
	xyz.re2	Mesh file
	xyz.par	Solver parameters
Driver	enrico.xml	Coupled driver settings
Submission	submit.lsf	Batch submission script

All workflow and runtime requirements use standard ECP and facility-supported libraries (e.g., Trilinos, HDF5).

Input Description and Execution

Problem Inputs To run ExaSMR coupled problems through the ENRICO driver, the following inputs are required:

- MC driver input (XML)
- MC geometry input file (XML or HDF5)
- MC material compositions file (HDF5)
- CFD parameter file (text)
- CFD mesh file (binary)
- CFD C++ user-defined functions (text)
- CFD OCCA user-defined kernels (text)
- CFD restart file (binary, optional)
- ENRICO driver input (XML)
- Batch submission script (text)

Table 34 summarize the collection of input files needed to execute a multiphysics simulation with ENRICO for the KPP-1 measurement case. The particular file names for the physics codes will be updated once available.

Resource Requirements

Estimated compute requirements for KPP-1 verification are two hours using the full Frontier or Aurora machines.

Problem Artifacts

Standard artifacts for coupled simulations include:

- Screen output of submitted jobs
- HDF5 output from MC solver
- NekRS state field output files (native binary format)

The NekRS files will be made available upon request; however, it is anticipated that these files will be very large (>1 TB). It is therefore not likely that the NekRS files will be useful for verification and we suggest that the other artifacts be used as the primary confirmation of execution. Because the run to be used for the FOM measurement will not be able to fully converge a coupled simulation, we cannot check the accuracy of computed results, but we can assess whether the simulation meets several sanity checks related to consistency of the numerical models and that computed quantities obey certain physical characteristics. These checks, all of which are printed to the screen output, include:

1. Verification of the consistency of the problem geometries. The ENRICO driver displays diagnostics related to volumetric mapping from the CFD mesh to the MC geometry, ensuring that the sum of the volume of the thermal/fluids elements matches the volume of the corresponding MC cell containing those elements. Exact agreement is not expected, but we anticipate the average volumetric error to be less than 1% and the maximum error for any element to be under 10% (some small regions may be slightly off due to modeling differences related to the gap between the fuel and clad in fuel pins). This check ensures that the physics models are geometrically equivalent and correctly aligned in space.

2. The fuel temperature should be nonuniform, i.e., the minimum and maximum temperatures should be different. The minimum temperature should be greater than the coolant inlet temperature of 531.15 K and the maximum temperature should be above this value. In a converged simulation, the fuel temperature is expected to reach a maximum of between 1000 K and 1200 K. For the FOM measurement, the actual temperature rise is expected to be much smaller, but the temperature should not exceed 1300 K.

3. The coolant/fluid temperature should be nonuniform, i.e., the minimum and maximum temperatures should be different. The minimum temperature should be approximately equal to the coolant inlet temperature of 531.15 K and the maximum temperature should be above this value. In a converged simulation, the coolant temperature is expected to reach a maximum of around 590–610 K. For the FOM measurement, the actual temperature rise is expected to be much smaller, but the coolant temperature should not exceed 620 K.

4. The eigenvalue (k_{eff}) computed by the MC solver should be approximately equal to 1. The actual value will depend on the temperature and coolant density distribution, but it is expected that it will likely be between 0.95 and 1.10.

KPP-1 FOM measurement data can be obtained by running provided ‘python‘ script to extract performance data from above artifacts:

```
>>> python process_fom.py case_name
```

This script will extract timing data for each of the physics solvers and evaluate the ExaSMR FOM metric. The MC transport FOM uses the follow information:

- the number of particle histories per eigenvalue cycle,
- the number of active cycles (the MC FOM is only taken over active cycles),
- the total time spent in the active cycles.

All of these quantities are contained in the HDF5 output file. The FOM calculation for the CFD solver uses the following information:

- the number of degrees of freedom in the problem,
- the number of time steps,
- the time spent in the CFD solve.

This information is contained in the ENRICO screen output and will be parsed by the provided python script.

Table 35: ExaSMR anticipated architecture support.

	Portability model	Aurora	Frontier
Shift	HIP	None	Optimized
OpenMC	OpenMP w/target offload	Optimized	Functional
NekRS	OCCA	Optimized	Optimized

4.3.2 Early Hardware Status

In this section, we describe the porting status for each of the physics codes within the ExaSMR project. Table 35 shows the expected level of architecture support for each code on the two exascale platforms. "Functional" indicates that a particular code is expected to be operational on a given platform, but no specific efforts are being taken to optimize the algorithms or implementation for that architecture. "Optimized" indicates that the code is expected to be performant on that platform.

Shift

Here we summarize the status of porting the Shift code to GPUs with the AMD HIP programming model. A more detailed description of this effort is available upon request. The port was performed by running the `hipify-perl` script on the existing Shift CUDA solver. This process was relatively smooth, requiring only a few manual corrections afterwards to handle certain CUDA features that don't exist in HIP, such as replacing the usage of `CUDART_NAN` with `nan("1")` and other similar constructs. Integrating HIP into Shift's build system was more of a challenge. HIP provides 3 pathways for integration into CMake:

1. legacy CMake variant in `$HIP_PATH/cmake`
2. modern CMake variant in `$HIP_PATH/lib/cmake/hip`
3. modern CMake variant using HIP as a primary language in `$HIP_PATH/lib/cmake/hip-lang`.

We used option (1) for our initial studies on Lyra in 2020; however we were aware that this was a short-term solution. Our current version of Shift uses a fully modern-CMake approach; thus, we have updated our build system to use HIP via option (3). So far, this approach is working on Spock with a few caveats that are described in **Build Details**.

Most of our effort thus far has been testing the performance of ROCm on the MI100 GPUs on Spock. From our initial HIP port on Lyra, the performance of Shift on AMD hardware has lagged the performance we have achieved on NVIDIA V100s. After some deep kernel analysis and optimization, we were able to achieve single node performance on Spock using ROCm 4.2 that meets or exceeds the V100 performance on our benchmark challenge problem. Unfortunately, the code performance resulting from ROCm 4.3 has not kept pace, and we are once again lagging the V100. These results and activities are summarized in 4.3.2–*Performance on Spock*.

Benchmark Problem and FOM

The reference problem for all of the following analysis is a three-dimensional SMR model with both fresh and depleted fuel configurations. We note that the depleted fuel configuration uses 1/4 symmetry and is our baseline challenge problem. The FOM performance measure that we are reporting is

$$\text{FOM} = \frac{\text{neutrons transported}}{\text{wall-clock time}},$$

in neutron/s.

In the performance plots that follow, we show FOMs for both *active* and *inactive* cycles as a function of number of particles per cycle. In these calculations Shift is performing an eigenvalue calculation using power iteration, and the inactive cycles represent the convergence of the eigenvector (source). During the active cycles quantities of interest such as neutron flux, reaction rates, and fission power are scored. Because the inactive cycles are only used to converge the source, no scoring is performed; thus, inactive cycles run

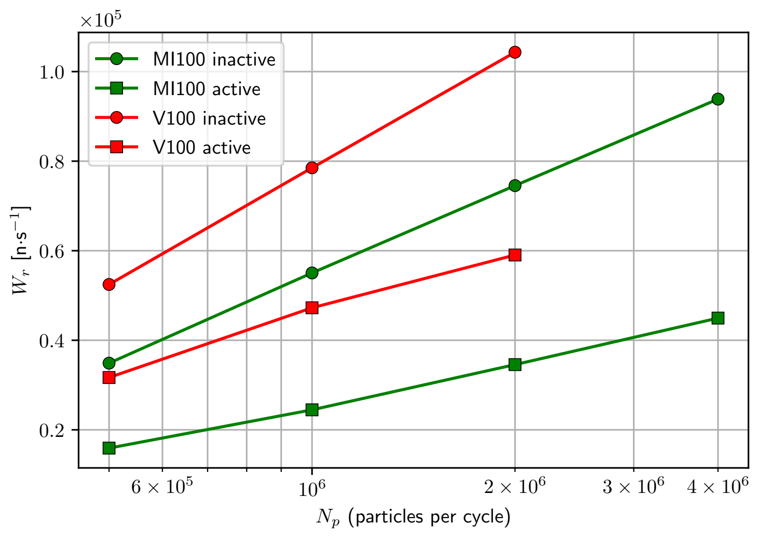


Figure 28: FOM for fresh fuel SMR problem.

at significantly lower cost (higher speed) than active cycles. Scoring significantly increase the algorithmic workload through additional cross section lookups, atomics, and non-uniform data access. In coupled reactor analysis problems, we general run many more active cycles than inactive cycles, so this is the more important metric.

Performance on Spock

All the tests that follow use a single MPI rank bound to a single GPU. Thus, we are comparing apples-to-apples GPU performance.

ROCm 4.2 The first test is a comparison between Summit (NVIDIA V100) and Spock (AMD MI100) on the fresh fuel SMR problem as shown in Fig. 28. Here we see that the performance of the MI100 lags the V100 by a factor of ≈ 2 in the active cycles. One benefit of the MI100 however is that it is capable of running more particles per cycle due to the increased memory on the GPU. On the V100 we can run 4 million particles per cycle, but we have to buffer the particle vector to fit the problem onto the device. For consistency, we only show results in which particle vectors are used without buffering.

In order to examine the roots of the performance differences we can look at the assembly files that are produced by AMD-Clang by giving the `--save-temps` command to the compiler. The resulting `-hip-amdgcn-amd-amdhsa-gfx908.s` assembly code can be parsed to find the number of vector registers (`vgpr`), the vector register spill count, and occupancy for each kernel. When we look at the GCN code produced on Spock and filter on kernels with register spill counts greater than 0 we get the results shown in Table 36. We note here that occupancy is the number of wavefronts per single instruction, multiple data (SIMD), and the maximum number per SIMD on the MI100 is 10. Also, each SIMD has a total of 256 `vgprs`.

Of the kernels in Table 36, the most performance critical are `dist_to_collision` and `calc_macro_rxn`. Also, as described above, `calc_macro_rxn` is only executed during active cycles. Nonetheless, what appears to account for the performance degradation between the GPUs is the large amount of register spills in the AMD code. The AMD compiler does not have anything equivalent to the `nvcc -maxrregcount=N` option that can be used to specify the number of registers to use per kernel. Instead, one is forced to explicitly specify `__launch_bounds__(MAX_THREADS_PER_BLOCK, MIN_WARPS_PER_EU)`. Because our MC transport algorithm uses a default block size of 256 for all kernels, setting this explicitly is simple. We also note that ROCm 4.2 uses a default setting of 1024 for `MAX_THREADS_PER_BLOCK` and 1 for `MIN_WARPS_PER_EU`. Thus, setting the size to 256 frees resources to use for registers at the potential expense of reduced occupancy. Setting the launch bounds for the `dist_to_collision` kernel to 256 gives 133 `vgprs`, 0 spills, and 1 occupancy. The performance using these settings is shown in Fig. 29 for the depleted fuel configuration. These results are significantly better

Table 36: Kernels in Shift with vector spill counts greater than 0.

Kernel	`vpgr`	`vpgr` spill count	occupancy
dist_to_collision	64	4734	4
sample_fission	64	8967	4
create_fission_sites	64	4171	4
process_collision	64	41087	4
calc_macro_rxn	64	32870	4
calc_macro_mt	64	2390	4
calc_partial_macro_rxn	64	14200	4
calc_partial_macro_mt	64	1894	4
calc_micro_rxn	64	14141	4
calc_micro_mt	64	1898	4
calc_micro_stateless	64	1823	4

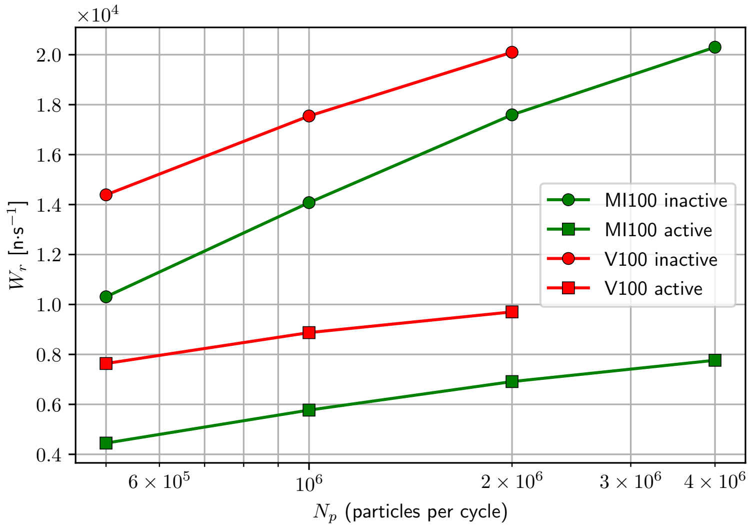


Figure 29: FOM for depleted fuel SMR problem with `__launch_bounds__(256)` set on the `dist_to_collision` kernel.

than the initial version with the MI100 at 2 M particles per active cycle only ~30 % slower than the V100.

A natural next step is to attempt to increase the occupancy of the kernels after removing the register spills. First, we apply `__launch_bounds__(256)` to all of the kernels in Table 36 to attempt to remove the spills. Analyzing those results, we can then toggle the second parameter (`MIN_WARPS_PER_EU`) to try and increase the occupancy without spilling any registers. In general we can increase this parameter in kernels that have a `vgpr` less than or near 128, since each SIMD only has 256 `vgprs`. Playing with the second parameter gives the results shown in Table 37. It required 2 compile/analyze iterations to get these settings. The performance on the depleted fuel problem with these launch bound parameters is shown in Fig. 30. At this point the performance gap between the V100 and MI100 has closed with the MI100 running less than 20% slower at 2 M particles during the active cycles. Furthermore, the performance of the MI100 during the inactive cycles is actually better than the V100 starting around 1 M particles per cycle.

The fact that the performance of the inactive cycles is better on the MI100 illuminates an idea, perhaps it is the reduced occupancy of the `calc_macro_rxn` kernel that is hampering performance during the active cycles. We know from our iterations on the launch bounds parameters that attempting to increase the occupancy of this kernel will result in register spills; however, that could be counter-balanced by the gains in

Table 37: Kernels in Shift with launch bound settings that result in 0 spills.

Kernel	threads/block	warps/EU	vpgr	vpgr spill count	occupancy
dist_to_collision	256	2	122	0	2
sample_fission	256	1	132	0	1
create_fission_sites	256	1	133	0	1
process_collision	256	1	146	0	1
calc_macro_rxn	256	1	133	0	1
calc_macro_mt	256	2	110	0	2
calc_partial_macro_rxn	256	2	102	0	2
calc_partial_macro_mt	256	1	128	0	2
calc_micro_rxn	256	2	101	0	2
calc_micro_mt	256	1	127	0	2
calc_micro_stateless	256	1	127	0	2

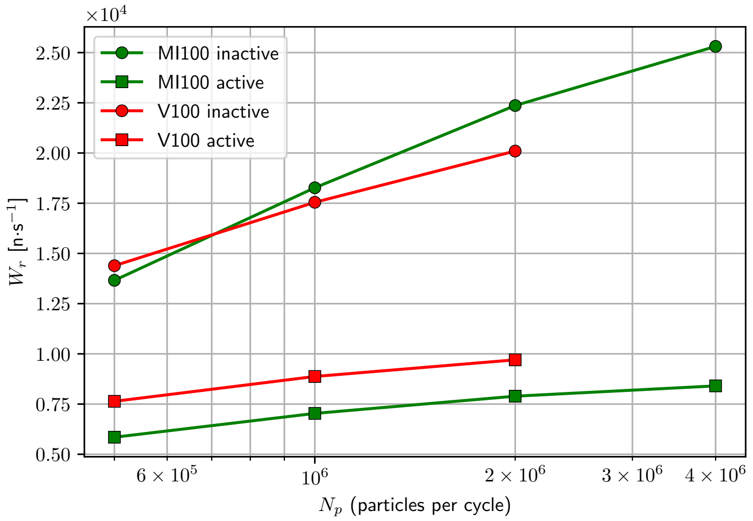


Figure 30: FOM for depleted fuel SMR problem using the launch bound parameters listed in Table 37.

Table 38: Kernels in Shift with launch bound settings that attempt to balance occupancy with register spills.

Kernel	threads/block	warps/EU	vpgr	vpgr spill count	occupancy
dist_to_collision	256	2	122	0	2
sample_fission	256	1	132	0	1
create_fission_sites	256	1	133	0	1
process_collision	256	1	146	0	1
calc_macro_rxn	256	2	128	316	2
calc_macro_mt	256	2	110	0	2
calc_partial_macro_rxn	256	2	102	0	2
calc_partial_macro_mt	256	1	128	0	2
calc_micro_rxn	256	2	101	0	2
calc_micro_mt	256	1	127	0	2
calc_micro_stateless	256	1	127	0	2

Table 39: Kernels in Shift using ROCm 4.3 with the same launch bound parameters from Table 38.

Kernel	threads/block	warps/EU	vpgr	vpgr spill count	occupancy
dist_to_collision	256	2	128	4341	2
sample_fission	256	1	142	0	1
create_fission_sites	256	1	139	0	1
process_collision	256	1	168	0	1
calc_macro_rxn	256	2	128	31456	2
calc_macro_mt	256	2	125	0	2
calc_partial_macro_rxn	256	2	113	0	2
calc_partial_macro_mt	256	1	128	0	2
calc_micro_rxn	256	2	112	0	2
calc_micro_mt	256	1	127	0	2
calc_micro_stateless	256	1	127	0	2

reduced latency through an increase in kernel occupancy. So, we toggle the launch bounds in this kernel yielding the results in Table 38. Now, for a small number of spills, 315, we have increased the occupancy by a factor of 2. The final performance results are shown in Fig. 31. Now, the MI100 meets or exceeds the V100 performance for all particle counts greater than 1 M for the active cycles. Additionally, the MI100 performance is better for the inactive cycles for all particle counts.

ROCm 4.3 Unfortunately, when we apply the launch bounds settings that yielded the performance results shown in Fig. 31 using ROCm 4.3 we see very different behavior. First, ROCm 4.3 gives very different register data than ROCm 4.2 as shown in Table 39. ROCm 4.3 yields higher register counts for nearly all kernels than ROCm 4.2. Also, we are now seeing large spill counts in both `calc_macro_rxn` *and* `dist_to_collision`. This has the resulting negative impact on performance shown in Fig. 32 in which the active cycles for ROCm 4.3 are 50% slower then the ROCm 4.2 version. The effect is even more dramatic on the inactive cycles.

We have since attempted some kernel reorganization to try and reduce the register pressure through code modification. Our first attempt was to make specializations for the total cross section evaluation performed in `dist_to_collision` through the addition of three additional kernels. This did remove the register spills from `dist_to_collision`, although it left `calc_macro_rxn` unchanged. Unfortunately, these changes did not result in any significant performance benefits as shown in Fig. 33. At this point, we are still trying to figure

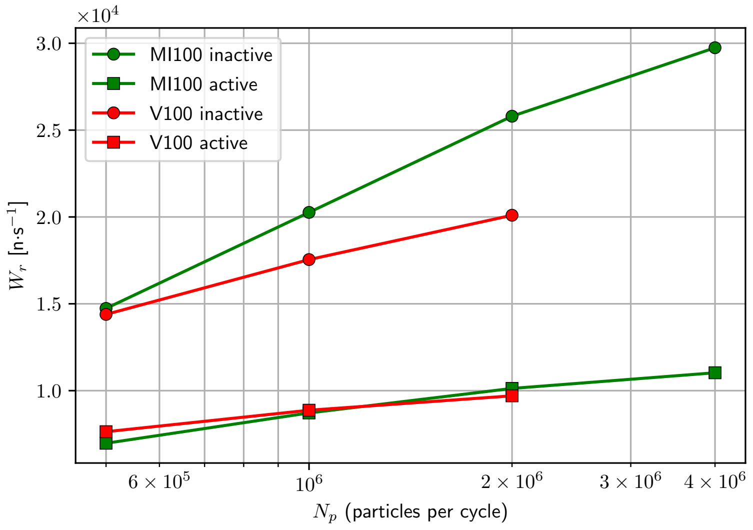


Figure 31: FOM for depleted fuel SMR problem using the launch bound parameters listed in Table 38.

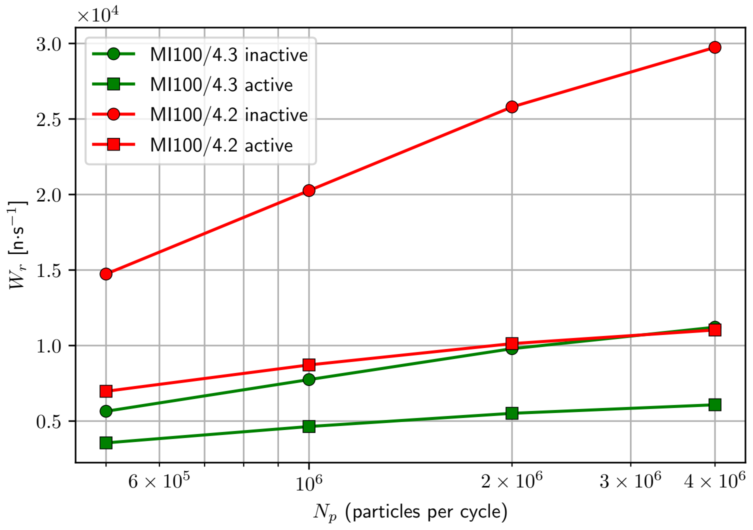


Figure 32: FOM for depleted fuel SMR problem using ROCM 4.3 and the launch bounds parameters from Table 39.

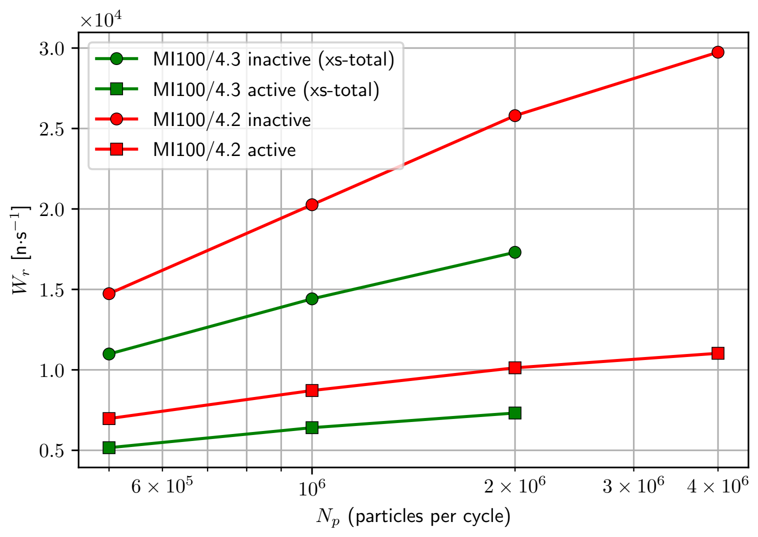


Figure 33: FOM for depleted fuel SMR problem using ROCm 4.3 and splitting the `dist_to_collision` into 4 kernels.

out different strategies for addressing the ROCm 4.3 performance, including getting AMD feedback on why the compiler behavior has changed so dramatically.

Build System and Compilation Issues

The head version of CMake, which will be released as 2.21, provides a new feature paid for by ECP that supports the use of HIP as a primary compiler. This allows the use of

```
enable_language(HIP)
```

in a modern-CMake variant build system. We have integrated these features into our build system and have used the head version to successfully build the HIP version of Shift on Spock. We note that all of our builds have used the GNU compiler environment. There are some items of note to point out here:

- We have been building our own version of Trilinos because there is no E4S [17] stack on Spock. Additionally, the current E4S configuration does not contain some of the Trilinos packages we need (Shards among others). Finally, there can be library version challenges, particularly with BLAS/LAPACK as described below.
- We have to explicitly include `hiprand` and `rocrand` in `HIPFLAGS`
- We are using the Hewlett Packard Enterprise (HPE) compiler environment (`CC`, `cc`, `ftn`) but still have to define the environment variables listed above and a separate set of `LDFLAGS`. This seems like something that should be included in the HPE compiler scripts.

We have encountered several AMD-Clang issues, most of which have been reported to AMD:

- `assert` does not currently work on device on AMD. This only affects our Design-by-Contract implementation in debug modes. We also have a work-around from AMD. Nonetheless, the work-around constrains the functionality across different systems (NVIDIA), or it forces us to maintain distinct code for different architectures.
- full debug builds (`-g -O`) fail with AMD-Clang,
- mixing static and shared BLAS/LAPACK libraries creates slurm runtime errors at problem completion due to incompatibilities in the libraries.

We have recently identified some potential roadblocks related to HIP and CMake moving forward. These can be summarized as follows:

1. HIP-nvcc is not well-supported. The latest versions of HIP do not configure/build cleanly with `HIP_PLATFORM=nvidia` and `HIP_COMPILER=nvcc`.
2. The HIP implementation in modern-CMake that we have works on AMD systems; however it appears to have a dependence on using AMD-Clang. At this point, it appears incapable of building device code on NVIDIA systems.

Both of these items are critical to the coding decisions we must make moving forward. We intended to use HIP as our NVIDIA/AMD platform-portability layer. We were able to build HIP versions of our code on Summit, using HIP-nvcc with a much older version of HIP. However, the integration between HIP-nvcc and HIP-Clang seems to be broken. Furthermore, the HIP language option in CMake does not currently support using HIP as a platform portability layer. The end result is *that HIP is quickly becoming an AMD surrogate language*, totally divorced from NVIDIA platforms.

We are currently working with CMake on the build issues with HIP, but we need more support from AMD to resolve the long-term support issues of HIP-nvcc.

OpenMC

Our primary goal is to prepare OpenMC for the Aurora supercomputer. We have made substantial progress toward this end in the past fiscal year. In particular, we have completed offloading several additional regions of OpenMC such that our overall porting effort is nearly feature complete. We have also had breakthroughs in terms of greatly improving the performance of OpenMC on GPU devices as compared to our initial runs in the Spring of 2021. OpenMC is now capable of running on an A100 GPU with performance roughly equivalent to 50–80 Xeon CPUs. We have also had success in compiling and running correctly on multiple GPUs from Intel as well as the MI100 GPU from AMD. However, efficient performance has so far only been achieved on the A100 GPU and when compiling with the LLVM compiler.

The OpenMP offloading port of OpenMC is now nearly feature complete. Recent months have seen the multipole cross section lookup routines and photon transport routines ported and run successfully on GPU. The only substantial capability not yet ported in OpenMC is tallying. While the tallying routines form a substantial body of code, we do not foresee any undue complexities in porting this region. Rather, we skipped this section so as to begin compiler evaluation, compiler debugging, compiler performance analysis, and OpenMC algorithmic performance optimization at the earliest possible stage. For this reason, all performance values given in this work refer only to inactive batch performance, as no tallies (other than global quantities like k_{eff}) are used.

While we are chiefly concerned with building and optimizing for Intel GPUs, we have recently been testing with a broad variety of compilers and GPU architectures. This has allowed us to make more rapid progress than if limiting development to a single compiler and architecture. Table 40 shows OpenMC's current build status with each of the three main vendors compilers as well as the open source LLVM compiler. While each of the GPU vendors' compilers only support a single architecture, the LLVM compiler is theoretically able to target both NVIDIA and AMD GPUs. However, a compile-time bug is currently preventing successful compilation using LLVM for AMD. Similarly, a compile-time bug is also preventing successful compilation with the NVIDIA NVHPC compiler for the A100, although thankfully the LLVM compiler does work for this architecture. We also note that there has been some thought from the LLVM team to possibly include support for Intel GPUs in the near future.

To generate a cross-architecture snapshot of OpenMC's current performance we ran a depleted pincell problem (featuring 272 nuclides) on several GPU architectures and recorded the inactive cycle calculation rate. The same test was also performed on a CPU node (a dual-socket Skylake Xeon 8180M node featuring 56 cores and 112 threads in total), which serves as a basis for comparison. GPU runs were performed using OpenMC's event-based mode, whereas the CPU runs were performed using OpenMC's traditional history-based mode. All configurations/architectures produced identical results owing to the efforts made to preserve reproducibility in OpenMC's event-based mode. For GPU tests, the maximum number of particles that can fit into each GPU's memory was used. For CPU runs, we did not observe a significant sensitivity on the number of particles per batch, so an identical number to what could fit on the A100 was used. The results

Table 40: OpenMC compiler status. “Yes” means that OpenMC can compile and run correctly in this configuration. “No” means there is a compiler bug that is currently preventing successful compilation and/or running of OpenMC in this configuration. “-” indicates that support is not intended for this combination.

	NVHPC	AOMP	Intel	LLVM
A100	No	-	-	Yes
MI100	-	Yes	-	No
Intel GPU	-	-	Yes	-

Table 41: Performance of OpenMC on a depleted pincell problem.

Node type	Compiler	Particles in-flight	Inactive batch calculation rate [particles/sec]	
			Pointwise	Multipole
AMD MI100	AOMP	650,000	3,882	N/A
NVIDIA A100	LLVM	825,000	90,043	67,377
2× Xeon 8180M (56c/112t)	LLVM	825,000	103,574	91,562

of our analysis, given in Table 41, reveal that reasonably efficient baseline performance has been achieved when compiling with LLVM for the A100. Conversely, the AOMP compiler does not appear to be producing efficient code for the MI100, so it will be important for us to interface with the AMD team going forward to identify issues in the AOMP compiler that are affecting OpenMC. Additionally, we ran into a bug in the AOMP compiler that prevented compilation of OpenMC when multipole cross section lookups were enabled. We also note that we collected data for an Intel GPU for this analysis, but due to NDA considerations that data is contained in the NDA appendix of this report. This appendix is available upon request.

NekRS

We have successfully ported NekRS on Frontier (Tulip, Spock) and Aurora (Iris, Arcticus) related architectures. NekRS uses the OCCA performance portability layer. A CUDA backend is used on NVIDIA GPUs, a HIP backend for AMD GPUs, and an OpenCL backend is used for Intel GPUs. Results for Intel GPUs are still governed by an NDA and are available upon request in a separate appendix.

The first problem is a canonical turbulent pipe flow problem. The performance results are provided in Table 42. For pipe flow, on Tulip GPU performance of a MI100 is roughly 85.5% of a V100 on Summit. Among the most affected parts of the solution algorithm is the assembly of the right hand side, which includes the non-linear terms (convective terms) and related de-aliasing and it is memory intensive. The kernels related to this part of the solver are 29.4% slower on a MI100 than a V100. This indicates immaturity and lack of optimization of the backend, including potential lack of optimization of compilers. A significant effort is underway to improve these numbers. Other key kernels, including the pressure and velocity solves are less affected (3.5–4.2 % slower) and are comparable in speed with the performance of the V100 on Summit.

The second problem is more relevant to ExaSMR challenge problems: a single subchannel model useful to study single GPU performance (“singlerod”). The performance results are provided in Table 43. Overall, the results are similar to the pipe flow problem, with an AMD MI100 GPU producing performance equivalent to about 85% of that produced by an NVIDIA V100 GPU.

4.4 MFIX-Exa

4.4.1 KPP Verification Plan

Verification Simulations

Table 42: NekRS baseline single GPU performance for turbulent pipe flow simulation with $Re = 19,000$, $E = 6840$, and $N = 7$. Quantity R is the ratio of throughput relative to Summit V100 (higher is better).

System	Device	API	$t_{step}(s)$	R
OLCF Summit	IBM Power9	C	1.99e+01	0.00427
OLCF Summit	NVIDIA V100	CUDA	8.51e-02	1.00
ALCF Theta-GPU	NVIDIA A100	CUDA	5.59e-02	1.52
HPE Tulip	NVIDIA V100	CUDA	8.85e-02	0.961
HPE Tulip	AMD MI60	HIP	1.41e-01	0.603
HPE Tulip	AMD MI100	HIP	9.96e-02	0.854

Table 43: NekRS performance on a single GPU. Simulations are performed for 500 steps and the averaged timing per step, t_{step}, is measured in seconds for 101–500 steps. The timestep size is Δt=1.2e-03 (CFL=7.3), $E = 7168$, $N = 7$, $n = 2.4\,\mathrm{M}$, $Re = 5000$. Quantity R is the ratio of throughput relative to Summit V100 (higher is better).

System	Device	API	t_{500}	t_{100}	t_{step}	R	ROCm
Summit	NVIDIA V100	CUDA	4.089e+01	9.018e+00	0.0795	1.00	-
ThetaGPU	NVIDIA A100	CUDA	2.503e+01	5.587e+00	0.0485	1.64	-
Spock	AMD MI100	HIP	5.007e+01	1.103e+01	0.0975	0.82	v4.2
Spock	AMD MI100	HIP	5.046e+01	1.110e+01	0.0982	0.81	v4.1
Tulip	AMD MI100	HIP	4.870e+01	1.060e+01	0.0953	0.84	v4.2
Tulip	AMD MI100	HIP	5.349e+01	1.160e+01	0.1045	0.76	v4.0

We will model National Energy Technology Laboratory (NETL)'s 50 kW chemical looping reactor (CLR) using MFIX-Exa, a fully coupled computational fluid dynamics-discrete element method (CFD-discrete element method (DEM)) code. If sufficient machine time is available, we will simulate 10 fluid time steps with subcycled particle steps.

The CLR geometry will be represented using AMReX embedded boundary (EB) and contain a fuel reactor, air reactor, cyclone, and loop seal. The components are connected by piping and the approximate dimensions of the fully assembled reactor are 4.2 m high, 1.1 m wide, and 0.22 m deep. A uniform grid size no larger than 1 mm will be used to discretize the valid regions of the domain.

The fluid will be modeled as a multicomponent ideal gas mixture comprised of methane (CH_4), hydrogen (H_2), steam (H_2O), carbon monoxide (CO), carbon dioxide (CO_2), oxygen (O_2), and nitrogen (N_2). Approximately 5 billion monodisperse spherical particles with a 200 µm diameter will be used to model the solids oxygen carrier. Particles will be treated as a mixture of hematite (Fe_2O_3) and wüstite (FeO).

The fluid and particle models will be coupled through the volume fraction field, interphase momentum transport (i.e., fluid-particle drag), interphase heat transfer via convective heat transfer, and interphase mass and energy transfer through heterogeneous chemical reactions.

A weak scaling study will be carried out on Summit–and, if sufficient machine time is available, replicated on Frontier–to establish GPU performance relative to CPU-only performance. We note that the fluid and particle physical properties will reflect those used in the challenge problem, however the scaling study will use a simple box geometry with periodic walls to maintain a uniform amount of work per process as the problem size is increased.

Simulation Artifacts

We will provide the simulation `inputs` file, constructive solid geometry (CSG) file, chemical reaction user defined function (UDF), job submission script, the standard output (screen file) from the simulation, and a brief write up summarizing the GPU-to-CPU performance analysis.

i. The `inputs` file is an ASCII text file containing keyword / value combinations that specify simulation parameters. The simulation domain size is given by `geometry.prob_lo` and `geometry.prob_hi`, the number of grid cells is specified by `amr.n_cell`, and the grid size is computed from these values. The filename of the constructive solid geometry file is defined by the `mfix.geometry_filename` keyword.

   Fluid and particle properties are specified using a series of keywords defining properties like viscosity and thermal conductivity for the fluid and collision model parameters for the particles. Namespaces for fluid and particle keywords and specified by the `fluid.solve` and `solids.types` keywords. The list of species making up the fluid is given by `fluid.species` while the list of species comprising particles is provided by the `solid0.species` keyword.

   Chemical reactions are specified using the `chemistry.solve` keyword, and for each reaction listed, the reaction stoichiometry is defined in terms of the fluid and particle species.

   The initial composition and temperatures of the fluid and particles are provided in the initial condition section of the `inputs` file as well as the particle diameter and initial density. Similarly, the boundary conditions section specifies the composition, pressure, temperature, and velocity of inflow boundaries and the pressure at outflow boundaries.

ii. The CSG file defines the CLR geometry and can be visualized using OpenSCAD.

iii. In addition to the stoichiometry provided in the `inputs` file, chemical reactions are incorporated into the simulation through user-defined functions in the `mfix_usr_reactions_rates.cpp` file. For each particle, fluid properties local to its position and its individual properties are passed to this file where an apparent reaction rate is computed in moles/sec for each defined reaction.

iv. The job submission script documents the requested compute resources.

v. The screen file captures runtime information including

   - number of MPI processes and GPUs utilized;

- total number of particles generated, and the number remaining after removing particles in contact with or outside the embedded boundary;
- maximum fluid velocity (by component) before and after the projection step;
- maximum and minimum fluid volume fraction;
- maximum and minimum fluid temperature;
- and a global mass balance.

The global mass balance in addition to the maximum and minimum fluid velocity, volume fraction, and temperature can be used to assess the reasonableness of the solution.

4.4.2 Early Hardware Status

Spock

All of the components essential to modeling the chemical looping reactor have been successfully tested, including:

- building and linking with the `csg-eb` library for inputting complex geometries,
- building and linking with hypre,
- incorporating heterogeneous chemical reactions,
- running a multi-GPU, multi-node simulation, and
- dual grid support for particle load balancing.

To demonstrate these capabilities, 25 time steps were taken for a system similar to the CLR fuel reactor. The setup, shown in Fig. 34, contains a 0.2032 m diameter cylinder centered in a rectangular domain with two internal obstructions used to mimic the loop seal dipleg (Fig. 34a right) and L-value outlet (Fig. 34a left). The geometry file was created using OpenSCAD and processed at run time by the `csg-eb` library to produce an implicit function which AMReX used to generate the embedded boundaries.

The fluid is a multicomponent ideal gas mixture comprised of CH_4, H_2, H_2O, CO, CO_2, O_2, and N_2, with an initial temperature of 1173 K and composition of 70% N_2 and 10% each CH_4, H_2, and CO. The system contains approximately 5M monodisperse spherical particles with a 620 µm diameter and initial composition of 80% Fe_2O_3 and 20% inert solids at a temperature of 1173 K. Three reduction reactions are included with artificially large reaction rates to guarantee observable conversion in the limited physical time simulated.

$$\begin{aligned} Fe_2O_3 + CH_4 &\rightarrow 8FeO + CO_2 + 2H_2O \\ Fe_2O_3 + H_2 &\rightarrow FeO + H_2O \\ Fe_2O_3 + CO &\rightarrow 2FeO + CO_2 \end{aligned} \tag{5}$$

The domain was decomposed into 16 ($2 \times 2 \times 4$) 64^3 grids with a uniform spacing of 2.5 mm. A second (dual) grid of $64 \times 64 \times 32$ cells was used for particle load balancing. The simulation was executed on the OLCF Summit and Spock systems using one GPU per MPI task, 16 GPUs total. The average time per step for the fluid, particle and coupling algorithms are shown in Fig. 35 with an approximate 3.5×, 5.8×, and 2.2× increase in time per advance for the fluid, particle, and coupling algorithms, respectively. The minimum, average, and maximum exclusive timings for the ten routines consuming the largest amount of run time on Spock are listed in Table 44 along with the timings for the same routines on Summit.

To compare single GPU performance of MFIX-Exa on Spock relative to Summit, a simple cube domain was constructed with periodic side walls, and inflow and outflow boundaries at the bottom and top faces. The reaction scheme in addition to the initial gas and particle compositions and temperatures are the same as the fuel reactor demonstration setup. The particle (DEM) and fluid-particle coupling algorithms were tested by varying the number of particles from 16^3 to 64^3, mimicking light and heavy GPU particle loads that are expected to occur because of differing solids concentrations found throughout the CLR. In each grid cell, 200 µm diameter particles are packed in a 4^3 cube lattice using a 2 µm spacing to ensure collision detection

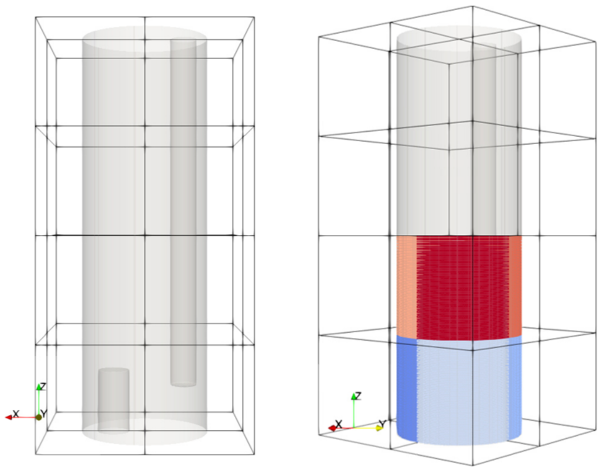


Figure 34: Illustration of fuel reactor demonstration unit. (a) Empty system showing internal obstructions and partitioning of system into 2 × 2 × 4 grids. (b) System with particles colored by initial MPI task.

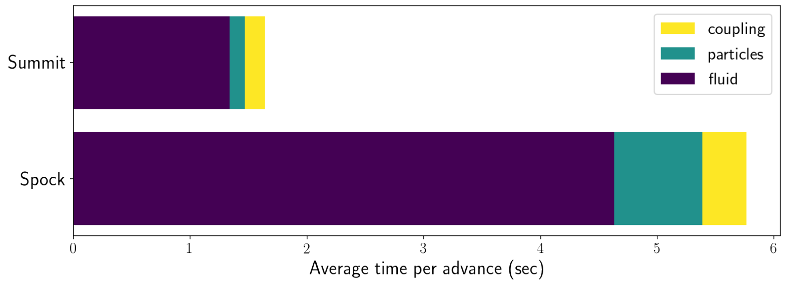
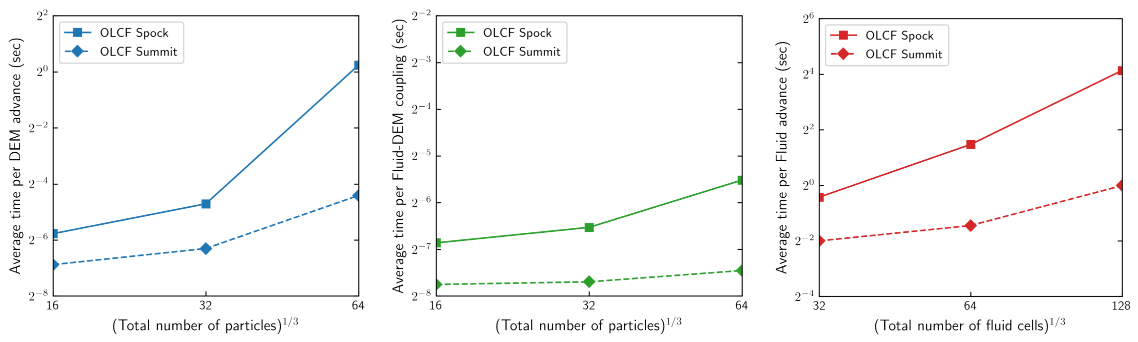


Figure 35: Average time per advance for the fluid, particle and fluid-particle coupling algorithms for the 16 GPU fuel reactor runs on Summit and Spock.

Table 44: The minimum, average and maximum exclusive timings for the ten routines consuming the most run time on Spock. Timing for the same routines are provided for the Summit run for reference.

	Summit			**Spock**		
Name	Excl Min	Excl Avg	Excl Max	Excl Min	Excl Avg	Excl Max
`ParallelCopy_finish()`	4.193	17.43	19.67	4.596	33.72	39.77
`HypreNodeLap::solve()`	0	1.09	8.722	0	2.378	19.03
`FillBoundary_finish()`	5.84	6.895	8	10.16	13.87	16.99
`HypreABecLap3::solve()`	0	0.7513	6.011	0	1.561	12.49
`FabArray::setVal()`	0.6203	0.6916	0.7739	10.31	11	11.83
`MLEBABecLap::Fsmooth()`	0.4639	0.5022	0.5395	8.595	8.817	9.053
`updateNeighborsGPU`	0.138	0.3552	0.5912	0.5939	4.611	8.807
`communicateParticlesStart`	0.8943	1.401	1.779	1.298	3.716	7.842
`DiffusionOp::ComputeLaphX`	0.7023	0.7637	0.8111	6.255	6.41	6.591
`amrex::packBuffer`	0.01673	0.1064	0.3182	0.01074	1.527	5.335



(a) **(b)** **(c)**

Figure 36: Single GPU performance on Spock and Summit.

and neighbor list generation and update algorithms are exercised fully. The average time per advance for the particle and coupling algorithms are shown in Figs. 36a and 36b, respectively. The fluid advance was similarly assessed by maintaining a constant particle load of 128^3 particles and domain length of 0.025,856 m and increasing the number of grid cells from 32 to 128 in each axial direction. The fluid advance results are shown in Fig. 36c.

Arcticus

We have successfully executed the single GPU MFIX-Exa benchmarks that include a triple-periodic box, fluidized bed and riser with periodic walls, a fluidized bed and riser in a cylindrical domain, and a simple hopper discharge. The benchmarks that utilize embedded boundaries were tested using both native AMReX implicit functions and the `csg-eb` library. Performance has been analyzed using an MFIX-Exa executable that was created using the Intel oneAPI SDK 2021.4 (2021.10.30.002) software stack on JLSE with implicit scaling disabled. Prior to this release, we had limited success with some tests running to completion while others would hang or unexpectedly crash.

4.5 WDMApp

4.5.1 KPP Verification Plan

Verification Simulations

On Frontier/Aurora, we will perform gyrokinetic simulation of the full-current International Thermonuclear Experimental Reactor (ITER) plasma to predict the height and width of the edge pedestal, which virtually determine fusion burn efficiency. The team's strategy will be to use WDMApp, which is focused on coupling the continuum code GENE and/or GEM in the core region and the particle-in-cell (PIC) code XGC at the edge. We will use 100 MW of alpha-particle heat source to maintain the core plasma profile and to provide the proper flow of heat to the edge code while the total-f edge code finds the steady-state pedestal height/shape and the divertor heat-flux width in the presence of neutral particle recycling at the wall.

The KPP FOM number will be obtained during the first short-time simulation period, at least 1/10 of the ion sound wave period (or at least 10 ion timesteps) after the initialization on 100% of the exascale platforms. In the base challenge problem, one ion species with mass 2.5 mH and real-mass electron species will be simulated. In the advanced challenge problem, the simulation will be initialized with a 50–50 % deuteron-tritium mixture, with multi-charge-state tungsten ions added. For the base challenge problem, it is expected that at least 24 cumulative wall-clock hours on 100% the available exascale platforms will be required for the pedestal to reach quasi-steady state. For the advanced challenge problem, we expect to use at least five cumulative days of 100% exascale wall-clock time.

Simulation Artifacts

Various macroscopic and microscopic artifacts will be produced and written in the ADIOSBP format produced through the ADIOS high-performance I/O library. Output files include the time-evolution of the pedestal profile, plasma rotation, divertor heat-load width, turbulence profile information, and the radial transport fluxes. ADIOSBP files can be converted to ASCII format via the ADIOS utility application BPLS. The EFFIS2.0 framework will be utilized to compose, execute, process and visualize the output data. The challenge science problem will be presented at major scientific conferences and written up for publication. The output data will be archived along with the performance numbers and visualizations, which will be made available to the community.

An ASCII FOM output file will be created that will contain the necessary information for FOM evaluation: the number of grid nodes, the number of particles used (by XGC), the number of the executed ion or electron timesteps—whichever has the smaller step size—the wall-clock time spent, and the number of compute nodes, without counting the initialization time. These numbers can be substituted into the WDMApp FOM definition formula and divided by the baseline FOM number to calculate the FOM enhancement factor. EFFIS will have a python script that evaluates the FOM and outputs the results in a human readable format. The base science challenge problem can be confirmed from the output files at various timesteps that contain the plasma-edge profile data and figures of the radial profiles for electron temperature, ion temperature, electron density, plasma rotation, turbulence, radial transport and divertor heat-load width.

4.5.2 Early Hardware Status

XGC is able to run and pass all the correctness tests on both Arcticus and Spock nodes making use of their GPUs. However, the tested performance using AMD GPUs are moderate to poor in comparison with a V100. On Spock, the poor performance of atomics has been confirmed as a contributor to overall moderate to poor performance. When the atomics are not used (in which case, the results are incorrect but the code structure and computations are identical), XGC's major GPU kernels perform 1–5× slower on a MI100 than on a V100. On the other hand, the most expensive kernel on V100, the electron push, is only 1.25× slower on MI100 even with atomics included. Thus, the effect of atomics is not uniform across the GPU kernels. The issue has been reported to the AMD and HPE engineers.

GENE has options to use sparse or dense LU solvers. The dense LU solver is more efficient on V100. GENE is able to run on Spock, with performance only 6% slower than on Summit if the sparse solver is used, with the same number of GPUs (3 Spock nodes versus 2 Summit nodes for 12 GPUs on each). But the performance is much worse when the dense solver is used. The slowdown of the dense field solver is because of the batched LU solver. The dense implementation on MI100 is extremely slow, so that the field solve is actually slower than on CPU. We have an open ticket with AMD about the dense LU solve issue. To achieve the performance improvement on Spock, we had to implement several workarounds for the performance limitations of the managed memory implementation in ROCm. Currently this is based on pinned host memory which can be accessed on GPU. GENE makes heavy use of managed memory, in both the initialization phase mostly from host and the main time loop accessed mainly from GPU. We have several strategies for further improvements. GENE is not yet running on Arcticus. Some work remains to port the last remaining CUDA/HIP kernels to use gtensor, a portability library, and to update the build system. Work is currently ongoing in concert with Intel engineers to port GENE to Arcticus.

GEM can be compiled on Spock, but errors occur when submitting a batch script. We suspect that the errors are due to inconsistencies in the floating-point types (float and double). This seems to be happening outside of OpenMP off-load regions. Additionally, it is likely that the interface parameter's type mismatches are related to the issue. The problem is still under investigation. We are trying to work with the HEP consultants on this issue.

ADIOS has been ported to Presque to allow the efficient writing and reading to the data analytics and optimizations (DAOs) storage system. EFFIS needs to be tested with DAOs for the future. On Spock we have ported EFFIS with XGC to ensure that the in-memory coupling can work on the Slingshot network; we are currently testing the coupling overhead on this system.

4.6 WarpX

Table 45: WarpX challenge problem details

Functional requirement	Minimum criteria
Physical phenomena and associated models	Dynamic Maxwell equations in the presence of a time-varying electromagnetic field with laser-driven and charged-particle source terms.
Numerical approach and associated models	PIC using finite-difference (i.e., FDTD) and/or and pseudo-spectral (i.e., FFT-based) analytical time domain temporal discretization with FFT-based field solve on AMR grids with solution in Lorentz-boosted frame.
Simulation details	Minimum of 10^{11} grid cells and 2×10^{11} macroparticles. Minimum number of five chained accelerator stages. Laser driver on the order of 1 PW peak.
Demonstration calculation requirements	Run a minimum of 100 timesteps so that performance is accurately measured by using a preloaded plasma column. Measure FOM for the FDTD and FFT-based solvers.

4.6.1 KPP Verification Plan

The exascale challenge problem is the modeling of a chain of tens of plasma acceleration stages. Realizing such an ambitious target is essential for the longer range goal of designing a single- or multi-teraelectron volt electron-positron high-energy collider based on plasma acceleration technology. The WarpX application uses AMReX for adaptive mesh refinement (AMR) and employs PIC methodology to solve the dynamic Maxwell equations to model the accelerator system.

The minimum completion criteria are designed to demonstrate that the project is on track toward modeling multi-teraelectron volt high-energy physics colliders based on tens to hundreds of plasma-based accelerator stages. The main goal is to enable the modeling of an increasing number of consecutive stages to reach higher final energy and to increase the precision of the simulations by performing simulations at higher resolutions in a clock time that is accessible. The goals are as follows:

- achieve a total plasma accelerator energy gain of at least 100 GeV by using five accelerator stages or more, and
- achieve a maximum wallclock time that is the same as the baseline.

The WarpX challenge problem details are listed in Table 45.

Verification Simulations

Facility Requirements

WarpX requires a recent version of CMake, version 3.22+ or the latest release are preferred for Exascale hardware. Analysis/post-processing/plot workflows require CPython 3.6+.

Input description and Execution

Problem Setup The physics problem is one stage of a laser-driven plasma accelerator with the following main physical parameters:

- **Plasma**: plasma density = $1.7 \times 10^{17}\,\mathrm{cm}^{-3}$; Length = 0.36 m.
- **Laser**: $a_0 = 1.7$; $w_0 = 50\,\mu\mathrm{m}$; Duration = 73 fs; $\lambda = 0.81\,\mu\mathrm{m}$.
- **Accelerated electron beam**: Charge = 0.15 nC; Width = 0.6 µm; Length = 3 µm; Emittance = 0.25 mm mrad.

and the following main numerical parameters:

- **Simulation box dimensions**: 161 µm × 161 µm × 161 µm.
- **Low-pass filter of source terms**: bilinear filter with 1 pass along x and y and 4 passes along z.
- **Current deposition algorithm**: Esirkepov method [18] with cubic macroparticles shape factor.
- **Field gathering algorithm**: Energy conserving method with cubic macroparticles shape factor.
- **Maxwell solver**: second-order leapfrog finite-difference method based on the Cole-Karkkainen-Cowan algorithm [19–21].
- **Particle pusher**: Vay method [22].
- **Time step limit**: $0.9999C$, where C is the Maxwell's solver Courant-Friedrich-Lewy (CFL) condition.
- **Boundary conditions (BC)**: periodic BC transversely and moving window longitudinally advancing at the speed of light.
- **Lorentz boost of the simulation frame**: 30.
- **Number of time steps**: 1000.
- **Number of plasma macroelectrons and macroprotons injected in the simulations per grid cell volume**: 1.

Floating numerical parameters that evolve with the size of the simulation:

- **the number of grid cells along each dimension**: N_x, N_y and N_z.

Problem Inputs All of the runtime parameters are given in a single inputs script (sample `inputs_4263` from the KPP run on 4263 Summit nodes of March 30, 2021), which is shown in Code Inputs 4.1 through 4.5.

Code Input 4.1: BOX PARAMETERS

```
max_step = 1000
# stop_time = 3.72930e-11
amr.n_cell = 5376 5376 29696
amr.max_grid_size = 128
amr.blocking_factor = 64
amr.max_level = 0
geometry.coord_sys   = 0
geometry.is_periodic = 1 1 0
# physical domain when running in the lab frame
geometry.prob_lo = -0.00016091428571428574 -0.00016091428571428574 -0.00016018285714285716
geometry.prob_hi = 0.00016091428571428574 0.00016091428571428574 0.0
```

Code Input 4.2: NUMERICS

```
warpx.verbose = 1
algo.current_deposition = esirkepov
algo.charge_deposition = standard
algo.field_gathering = energy-conserving
algo.particle_pusher = vay
algo.maxwell_solver = ckc
interpolation.nox = 3
interpolation.noy = 3
interpolation.noz = 3
warpx.use_filter = 1
warpx.filter_npass_each_dir = 1 1 4
warpx.cfl = .9999
warpx.do_pml = 0
warpx.do_dynamic_scheduling = 1
warpx.algo.load_balance_intervals = -1
# Moving window
warpx.do_moving_window = 1
warpx.moving_window_dir = z
warpx.moving_window_v = 1.0 # in units of the speed of light
```

Code Input 4.3: BOOST PARAMETERS

```
warpx.gamma_boost = 30.0
warpx.boost_direction = z
```

Code Input 4.4: PLASMA

```
particles.nspecies = 3
particles.species_names = electrons ions beam
particles.use_fdtd_nci_corr = 1
particles.rigid_injected_species = beam

electrons.charge = -q_e
electrons.mass = m_e
electrons.injection_style = NUniformPerCell
electrons.num_particles_per_cell_each_dim = 1 1 1
electrons.momentum_distribution_type = "gaussian"
electrons.xmin = -150.e-6
electrons.xmax =  150.e-6
electrons.ymin = -150.e-6
electrons.ymax =  150.e-6
electrons.zmin = -1.
electrons.zmax = 0.383
electrons.profile = constant
electrons.density = 1.7e23
electrons.do_continuous_injection = 1

ions.charge = q_e
ions.mass = m_p
ions.injection_style = NUniformPerCell
ions.num_particles_per_cell_each_dim = 1 1 1
ions.momentum_distribution_type = "gaussian"
ions.xmin = -150.e-6
ions.xmax =  150.e-6
ions.ymin = -150.e-6
ions.ymax =  150.e-6
ions.zmin = -1.
ions.zmax = 0.383
ions.profile = constant
ions.density = 1.7e23
ions.do_continuous_injection = 1

beam.charge = -q_e
beam.mass = m_e
beam.injection_style = "gaussian_beam"
beam.x_rms = 6.e-7
beam.y_rms = 6.e-7
beam.z_rms = 3.e-6
beam.x_m = 0.
beam.y_m = 0.
beam.z_m = -107.e-6
beam.npart = 50000
beam.q_tot = -1.5e-10
beam.momentum_distribution_type = "gaussian"
beam.ux_m = 0.
beam.uy_m = 0.
beam.uz_m = 1956.9469069265976
beam.ux_th = 0.4166666666666667
beam.uy_th = 0.4166666666666667
beam.uz_th = 39.138943248532286
beam.zinject_plane = 0.e-6
beam.rigid_advance = true
beam.projected = true
beam.focused = false
```

Code Input 4.5: LASER

```
lasers.nlasers     = 1
lasers.names       = laser1

laser1.profile      = Gaussian
laser1.position     = 0. 0. -1.e-6 # This point is on the laser plane
laser1.direction    = 0. 0. 1.      # The plane normal direction
laser1.polarization = 0. 1. 0.      # The main polarization vector
laser1.e_max        = 6.82274e12       # Maximum amplitude of the laser field (in V/m)
laser1.profile_waist = 50.e-6       # The waist of the laser (in meters)
laser1.profile_duration = 7.33841e-14   # The duration of the laser (in seconds)
laser1.profile_t_peak = 1.467682e-13    # The time at which the laser reaches its peak (in seconds)
laser1.profile_focal_distance = 0.00875  # Focal distance from the antenna (in meters)
laser1.wavelength = 0.8e-6         # The wavelength of the laser (in meters)
laser1.do_continuous_injection = 0
```

Estimate of resource allocation

Runtimes from the KPP runs that were performed on March 30, 2021, using weak scaling from 320 to 4263 nodes on Summit, range from 553 to 601 seconds for 1000 time steps. Current testings of WarpX on AMD MI200 show runtimes of about 40% the runtimes observed on NVIDIA V100. While we do not know the exact number of nodes that will be on Frontier, conservatively assuming that we can at least match the walltime performance of Summit, we should be able to run at least 4 full scale and more smaller simulations at lower scale in one hour of machine time, assuming 1000 time steps per simulation. More simulations can be run if cutting on the number of time steps, with a minimum of 100 time steps per simulations for proper statistics per run.

Estimate of resource allocation

Runtimes from the KPP runs that were performed on March 30, 2021, using weak scaling from 320 to 4263 nodes on Summit, range from 553 to 601 seconds for 1000 time steps. Current testings of WarpX on AMD MI200 show runtimes of about 40% the runtimes observed on NVIDIA V100. While we do not know the exact number of nodes that will be on Frontier, conservatively assuming that we can at least match the walltime performance of Summit, we should be able to run at least 4 full scale and more smaller simulations at lower scale in one hour of machine time, assuming 1000 time steps per simulation. More simulations can be run if cutting on the number of time steps, with a minimum of 100 time steps per simulations for proper statistics per run.

Simulation Artifacts

The main artifact that is produced during the simulation is a single ASCII file that contains:

- Number of MPI domains and GPUs.
- Version of libraries (AMReX and PICSAR) and source code of WarpX.
- Time step value and grid cell sizes.
- Elapsed CPU time elapsed since the start of the simulations at each time step.
- Automated builtin timing of main individual subroutines.
- Warnings.

A sample from output file `output_4263.txt` (from KPP run on 4263 Summit nodes of March 30, 2021) is shown in Code Output 4.1.

Code Output 4.1: WarpX Output

```
Initializing CUDA...
CUDA initialized with 1 GPU per MPI rank; 25578 GPU(s) used in total
MPI initialized with 25578 MPI processes
MPI initialized with thread support level 3
AMReX (21.03-68-g92945ad3a356) initialized
particles.nspecies is ignored. Just use particles.species_names please.
lasers.nlasers is ignored. Just use lasers.names please.
WarpX (21.03-79-g106a228725cc)
PICSAR (b35f07243c51)
Level 0: dt = 1.9966466e-16 ; dx = 5.986394558e-08 ; dy = 5.986394558e-08 ; dz = 3.235553937e-07

Grids Summary:
  Level 0   409248 grids  858255261696 cells  100 % of domain
            smallest grid: 128 x 128 x 128  biggest grid: 128 x 128 x 128


STEP 1 starts ...
STEP 1 ends. TIME = 1.9966466e-16 DT = 1.9966466e-16
Walltime = 1.223000876 s; This step = 1.223000745 s; Avg. per step = 1.223000876 s

...

STEP 1000 starts ...
re-sorting particles
STEP 1000 ends. TIME = 1.9966466e-13 DT = 1.9966466e-16
Walltime = 601.3638601 s; This step = 0.728943904 s; Avg. per step = 0.6013638601 s
Total Time                     : 602.7832846

TinyProfiler total time across processes [min...avg...max]: 602.8 ... 602.8 ... 602.9

...

Total GPU global memory (MB) spread across MPI: [16128 ... 16128]
Free  GPU global memory (MB) spread across MPI: [6158 ... 10436]
[The         Arena] space (MB) allocated spread across MPI: [12096 ... 12096]
[The         Arena] space (MB) used      spread across MPI: [0 ... 0]
[The  Device Arena] space (MB) allocated spread across MPI: [8 ... 8]
[The  Device Arena] space (MB) used      spread across MPI: [0 ... 0]
[The Managed Arena] space (MB) allocated spread across MPI: [8 ... 8]
[The Managed Arena] space (MB) used      spread across MPI: [0 ... 0]
[The  Pinned Arena] space (MB) allocated spread across MPI: [221 ... 366]
[The  Pinned Arena] space (MB) used      spread across MPI: [0 ... 0]
AMReX (21.03-68-g92945ad3a356) finalized
```

The main output number is the wall time at time step 1000, illustrated in blue in Code Output 4.1, which is approximately 601 seconds in this example.

Lightweight I/Os may also be turned on that may include in situ calculation of accelerated electron beam moments and total electromagnetic field energy (history of moments and field energy saved in an ascii file), and two-dimensional (slice) snapshots of field data for post-processed rendering.

4.6.2 Early Hardware Status

Spock

Equipped with AMD MI100 GPUs, Spock has provided a testbed for Frontier since mid-2020. WarpX compiles and runs for the provided hardware and continuous development updates are covered by compilation tests in CI. However, the performance falls short of that expected from Summit, at least in double precision. This is illustrated in Table 46, which compares the performance of WarpX in both double and single precision between AMD MI100 and NVIDIA V100 hardware on a standard warm plasma benchmark problem with periodic boundary conditions. While performance on the two cards is very similar in single precision, in double precision WarpX is about 2.5× slower on the MI100 than V100. The reason for this discrepancy is the lack of hardware acceleration for double-precision floating-point atomic adds on MI100. The lack of this feature is especially noticeable on this test problem, which uses 8 particles per cell and Esirkepov deposition with 3rd order particle shapes to particularly stress the current deposition kernel in WarpX. We note this problem has been resolved on MI200 hardware (see § 4.6.2–*Crusher*).

In order to assess the scalability of the network stack on Spock, a weak-scaling study using the KPP

Table 46: Performance comparison between MI100 and V100, in single and double precision, on a warm plasma benchmark. Times are in seconds.

	MI100	V100
Single	38.59	37.39
Double	128.54	50.17

Table 47: Weak scaling study using the KPP problem setup on Spock

$N_{\texttt{GPU}}$	$N_x \times N_y \times N_z$	time/step (s)
1	$256 \times 256 \times 768$	1.2646
2	$512 \times 256 \times 768$	1.2786
4	$512 \times 512 \times 768$	1.3049
8	$512 \times 512 \times 1536$	1.3112
16	$1024 \times 512 \times 1536$	1.3279

problem setup has been performed. The results are shown in Table 47. The base size for this scaling study used the inputs file shown in § 4.6.1 above, scaled down to a size of $256 \times 256 \times 768$ cells so that the problem fits on 1 GPU. This run was then scaled up by doubling the domain size in either the $x-$, $y-$, or $z-$ direction and also running on twice as many GPUs. Under perfect weak scaling, the run time should remain constant. The actual run time, however, increased by about 5%, owing to an increased number of communicating next-neighbor GPUs when exchanging guard cells and particles between MPI domains. The full communication pattern for 3D domain decomposition would require at least $3 \times 3 \times 3 = 27$ GPUs and should then, for higher GPU counts, not increase the average communication cost per participating rank any further. The scaling study was conducted on up to 4 nodes (16 GPUs), the largest run available to us under the non-CAAR ECP access to Spock. Thus, to the extent that we have been able to test it, the MPI performance on Spock appears to be quite good on our challenge problem.

All performance results listed in this section were collected using ROCm 4.3. We have noted some performance differences between ROCm 4.1 and 4.3 (we cannot use 4.2 due to a compiler bug) on Spock. In particular, current deposition in faster in 4.3, but the `GatherAndPush` kernel, which collects electric and magnetic field data at the particle positions and updates their positions and momenta, is actually slower in the newer version of the software. The reason appears to be an increased tendency towards register spilling in ROCm 4.3 compared to 4.1. We understand that other application development teams have noticed this regression as well and reported it to AMD via the COE webpage. While we are exploring workarounds (for example, manually limiting the number of threads per block for kernels that spill), we are eager to see this fixed in new versions of ROCm. Because of the different effects on the deposition and field gathering kernels, the warm plasma example in Table 46 is faster with ROCm 4.3, but the KPP examples in Table 47 are all about 10% faster with 4.1.

Crusher

The next-generation testbed for Frontier, which is equipped with MI200 GPUs, was not available before the review report deadline. Nonetheless, WarpX was tested on a single MI200 GPU through a collaboration with AMD engineers. Table 48 compares the performance of WarpX on a variety of GPUs, including MI200. AMD's MI200 are, similar to NVIDIA's K80 GPUs, integrating two GPUs on one board and WarpX uses one MPI rank per GPU. Thus, we have shown results using both "half" of an MI200 GPU, which, like the other runs in the table, used only 1 MPI rank, and using the "full" MI200, which was achieved by running the same inputs script with two MPI ranks. These performance numbers were gathered using the "warm plasma" benchmark from § 4.6.2–*Spock* above. We have also included performance results using NVIDIA's A100 hardware, collected on NERSC's Perlmutter platform.

There are two points to make about these numbers. The first is that the issue with native support for double precision atomic adds discussed in § 4.6.2–*Spock* seems to be resolved on MI-200. Even using only half of the dies on a MI-200 GPU, the current deposition time for MI-200 beats that of NVIDIA's AI100 hardware.

Table 48: Performance comparisons between various GPU hardware using WarpX's warm plasma benchmark. All runs used double precision.

GPU	Total time (s)	Deposition (s)	GatherAndPush (s)
MI100	116.4	67.5	40.2
MI200 (half)	37.3	21.1	7.4
MI200 (full)	20.5	11.0	4.0
V100	48.6	34.3	6.8
A100	33.1	24.1	4.1

The second point is that on MI-200, the GatherAndPush kernel still lags behind the NVIDIA GPUs. We believe that this is due to the register spilling issue also discussed in § 4.6.2–*Spock*. In our experiments, eliminating the register spill by launching the GatherAndPush kernel with only 64 threads per block improves the runtime of that kernel on this benchmark by about 40% on MI200.

Software Readiness

WarpX tests all changes to the code base with continuous integration (CI). Compilation tests using the latest public AMD ROCm and Intel oneAPI compilers ensure that contributed code does not break capabilities for Exascale hardware. WarpX's Spack packages `warpx` and `py-warpx` are integrated in E4S and build variants via the `warpx compute=<hip/sycl/cuda/omp>` options allow selecting the targeted architecture. Compilation on HPE/Cray machines such as Spock and Perlmutter with CMake required an extension of the `FindMPI.cmake` package by the WarpX developers for the CMake 3.22 release, ensuring that Cray compiler wrappers can be interrogated for MPI flags, even if early-access AMD, Intel or NVIDIA compilers are used directly.[2] Furthermore, mixed-language compilation problems in AMD's HIP CMake packages were triaged and merged to ROCm 4.4.[3]

5. EARTH AND SPACE SCIENCE APPLICATIONS

End State: Deliver a broad array of comprehensive science-based computational applications able to provide, through effective exploitation of exascale HPC technologies, breakthrough modeling and simulation solutions to fundamental issues and scientific questions centered on key planetary processes and the origin of the universe.

The Earth and Space Science Applications (ESS) application L3 area (Table 49) spans fundamental scientific questions from the origin of the universe and chemical elements to planetary processes and interactions affecting life and longevity. These application areas treat phenomena where controlled and fine resolution data collection is extremely difficult or infeasible, and in many cases fundamental simulations are the best source of data to confirm scientific theories and predict critical phenomena.

The key objective in the area of ESS is to utilize exascale resources to carry out simulations of phenomena with massive ranges of space and temporal scales, and where controlled data collection is extremely limited or impossible. These applications are critical to mankind's well-being and understanding of fundamental questions of the universe, and in many cases advanced simulation is the most effective vehicle for gaining insight into these processes. As their computing requirements are massive, it is critical to develop these codes to make efficient use of exascale computing resources.

5.1 ExaStar

5.1.1 KPP Verification Plan

Verification Simulations

[2] https://gitlab.kitware.com/cmake/cmake/-/merge_requests/6264
[3] https://github.com/ROCm-Developer-Tools/HIP/pull/2280

Table 49: Summary of supported ESS L4 projects.

WBS number	Short name	Project short description	KPP-X
2.2.3.01	ExaStar	Demystify the origin of chemical elements	KPP-2
2.2.3.02	ExaSky	Cosmological probe of the Standard Model	KPP-1
2.2.3.03	EQSIM	Seismic hazard risk assessment	KPP-1
2.2.3.04	Subsurface	Carbon capture, fossil fuel extraction, waste disposal	KPP-2
2.2.3.05	E3SM-MMF	Regional assessments in earth systems	KPP-1

The ExaStar challenge problem to be run on Frontier is a 3D simulation of the initial stages of evolution following the collapse and iron core bounce of a core-collapse supernova (CCSN). The calculation will exercise the integrated multi-physics of the code, including gravity, hydrodynamics, nuclear reactions, and neutrino transport.

Initial Conditions

The progenitor star model will be chosen at runtime from the best available models. The most likely progenitor models are: (1) the solar metallicity 12 solar mass progenitor of Sukhbold, Woosley, and Heger [23] (modeled using the spherically symmetric stellar evolution code Kepler) which is believed to represent a common type of massive star that produces a CCSN; (2) the binary stellar merger model of Menon and Heger [24], chosen because it is believed to closely mimic the progenitor system of SN 1987A, the only CCSN from which there are multi-messenger signals to date.

Domain and Resolution

The physical domain of the simulation will extend from the center of the star outward to fully enclose the helium shell of the evolved star. The precise location of this radius is progenitor-dependent, but it is always more than 10,000 km. The highest spatial resolution enabled with AMR will be at least 1 km at the surface of the proto-neutron star (i.e., in approximately the inner 100 km of the event). The spectral energy distribution of neutrinos of all types (i.e., electron, mu, tau, and their anti-particles) will be resolved using 20 energy groups spanning the range 0 to 300 MeV.

Included Physics

Gravity Gravity will be treated in Newtonian form (with a correction factor to the potential to approximate general relativistic modifications) using a multipole approach with up to 12 moments. ExaStar has also developed a full general relativistic (GR) spacetime solver and GR magnetohydrodynamics module which will be used instead if ready. However, these are considered Stretch Problem goals.

Neutrino transport Neutrinos will be treated using a multi-frequency two-moment formulation of the transport equation solved using Discontinuous Galerkin methods. Different flavors of neutrinos and their antiparticles will be included. An extensive set of tabulated neutrino-matter interaction rates—which include emission, absorption, scattering, and pair production from various nuclear and nucleonic processes-—-will be used. The two-moment method requires specification of closure relations describing the angular distribution of the radiation field, for which analytic closures will be applied. ExaStar is developing multi-angle Boltzmann transport capabilities using MC methods, and these could potentially be used to evaluate the accuracy of the closure relation or be integrated into the simulation to provide real-time calculation of closures. The MC capabilities, however, are considered a stretch goal.

Equation of State The simulation will use a tabulated high-density equation of state (EOS) that will provide pressures, entropies, and all other required thermodynamic values as required by, for example, the hydrodynamics. The EOS will be constructed so that it is consistent with the neutrino interaction rates. As the high-density EOS is not entirely understood from first-principle physics, various proposed forms for the EOS have been developed. The available set of coupled rates and EOS tables in the code will include, at a minimum, the commonly used SHF0 EOS, which is the likely choice to be used for the challenge problem.

Simulation Artifacts

Input Files

The simulation inputs will consist of progenitor star model files (ASCII format, as they are all spherically symmetric) and input compile-time and run-time parameter files which will specify the physics modules used in the calculation, and aspects of the simulation such as domain size and AMR refinement criteria.

Output Files

The simulation code periodically outputs “restart” files that contain essentially the entire state of the simulation at the specific time. More frequently, the code can output smaller “plot” files which contain only the mesh information and a reduced description of the system state, including the thermodynamic variables and relevant integrated quantities of the neutrino radiation field (e.g., luminosity, energy density). For our challenge problem simulation, we will generate between 10 and 20 “restart” files (in HDF5 format) which will track the evolution of the system from the initial to the final state. In addition, we will output roughly 200 “plot” files, which will provide a more highly time resolved description of the same evolution.

Log Files

The code produces a continuously updated log file that includes profiling information, AMR activity, I/O activity, and many other runtime quantities. A global conservation file is also appended to at every time step and is used for monitoring energy, mass, and momentum conservation, among other conserved quantities. A job log file will include a list of nodes of Frontier that were allocated.

Visualizations and Analytics

The data in the plot files can be used to construct images and movies of the evolution of the system. We anticipate producing visualizations of several relevant system variables, such as the neutrino radiation field, the matter density and entropy fields, and the isotopic composition throughout the domain. This will be done via post-processing, using either VisIT or Paraview. Static visualizations (such as image slices and volume renderings) will be generated for detailed quantities like the radiation field and composition, which are only available in the restart files. A limited number of such sets of images will be produced for the simulation (corresponding to times when restart files are produced). The matter density and entropy fields, which are available in the plot files, can be visualized with much higher frequency and will be used to generate animations. Plots showing global quantities, such as explosion energy versus or shock radius versus time, may be constructed from the plot files to assess whether the simulation is behaving in a physical manner.

KPP Validation

The runtime artifacts will confirm the parameters stated in the challenge problem description were used. The log file will contain information about the total number of MPI ranks and the memory per rank used. The job log file will include a full list of nodes on Frontier that were allocated. Our overall performance will be compared on a per timestep basis versus a smaller run that does not marshal the GPUs.

5.1.2 *Early Hardware Status*

Initial efforts to compile and run pieces of the multi-physics simulation code on Spock have been made, though progress is still limited. Execution of the Flash-X code suite (the primary multi-physics simulation tool to be used for our exascale challenge problem) has been hung up by compiler errors related to using OpenMP with Fortran. The issue has been reported to HPE, but as yet no fix has been implemented. As risk mitigation, ExaStar has also been integrating physics capabilities into the multi-physics code framework Castro, which was natively built on AMReX, and for which most Fortran code pieces have now been converted to C++. Compilation of Castro was initially held up due to the lack of support for C++17 in the device compilers. A workaround has been found, and Castro now has been compiled for test problems. The AMReX code framework itself has been used extensively on Spock. Once the issues related to Fortran compilers are addressed, we intend to run standard test problems and assess performance.

5.2 ExaSky

5.2.1 KPP Verification Plan

The ExaSky project supports two large-scale cosmological simulation codes, HACC (Hardware/Hybrid Accelerated Cosmology Code) [25, 26] and Nyx [27, 28]. Both codes use (different) N-body methods for gravity in an expanding universe; gas dynamics in HACC is treated via a higher-order CRK-SPH method (Conservative Reproducing Kernel-SPH [29]), whereas Nyx uses a block-structured Eulerian AMR technique. ExaSky is a KPP-1 project with an FOM ratio requirement and proof of challenge problem readiness both in terms of the targeted science coverage and overall performance. The KPP-1 requirements apply to runs with HACC; Nyx provides an essential complementary capability for code accuracy and subgrid model verification.

FOM Ratio

The FOM ratio requirement is to be met by the HACC cosmological simulation code running in both gravity-only and gas dynamical modes on Frontier and Aurora. The FOM ratio has already met the threshold requirement (FOM ratio > 50) on pre-exascale systems; currently reported numbers are 64.2 (measured) and 72.2 (extrapolated) on Summit. The FOM ratio computation is carried out by running a scaled-down version of the challenge problem (already carried out on the chosen baseline system, in our case, Theta (an Intel KNL system at ALCF), and then measuring throughput in two modes: gravity-only and gravity + hydro. The results are combined to yield a single FOM:

$$\text{FOM} = \sqrt{\left(\frac{n_p^3}{t}\right)_{grav}\left(\frac{2n_p^3}{t}\right)_{hydro}} \tag{6}$$

where n_p is the number of simulation particles per dimension (in the case of the hydro run, there are two species with the same number of particles) and t is the time taken for a time-step at a given fiducial moment of the simulation. The advantage of this FOM definition is that it is simple, yields a single number, and requires no correction factors when comparing results from two systems (taking ratios of the respective FOMs) as long as the two FOM runs were done with the same code parameters (force and mass resolution, etc.). Note that the number of particles can and will be different, as larger problems can be run on the exascale machines, with an increased FOM ratio, as expected, provided weak scaling continues to hold.

The problem with factoring the complexity of subgrid models into the FOM is that since these models are continuously evolving and were not run – and will not be run—on systems such as Mira and Titan, it is very difficult to estimate what the performance ratios would be across platforms, especially if the baseline platforms are either not available or it is not worth porting new applications to obsolete systems (for which they would not be optimally written in any case). Fortunately, we can essentially eliminate the subgrid model performance from the FOM for two reasons: 1) the subgrid models are entirely local, and as such do not affect the weak scaling performance of the code; 2) the primary effect of having subgrid models is twofold, an increase in the time for an individual short-range computational map that combines gravity and hydrodynamic forces, and a reduction in the actual value of the short-range timestep (higher time resolution) once the subgrid models are incorporated. The latter effect can be quite significant, leading to increases in the overall number of timesteps by one or two orders of magnitude, whereas the actual increase in time due to the additional work per timestep is only of order 10-20%. For this reason, the FOM remains controlled by the time spent in the main solvers, i.e., gravity + hydro, and therefore, the performance of these can be used to determine this value without having to consider how the new subgrid models would perform on the baseline platform(s).

Table 50 summarizes the current status of our FOM measurements as reported in the ECP FOM tracker on Jira. The baseline simulation was carried out on Theta and two major FOM measurements then followed on Summit in 2019 and 2020. In 2019, we reached an FOM just over 20. In the following year, the performance of CRK-HACC was improved considerably and we reached an FOM of more than 50. We also demonstrated excellent weak scaling on Summit for both gravity and hydro solver. We also carried out a

Table 50: ExaSky FOM improvement over time. The baseline run was carried out on Theta. We report the node count for the actual FOM runs, however, following the ECP convention, the FOM_{time} has been scaled up to 4392 Theta nodes and 4600 Summit nodes to ensure that all ECP projects are compared on the same footing. Column 7 reports the measurement on each machine following Eq. 6 (including the extrapolation) and Column 8 reports the ratio with respect to the Theta baseline measurement. We have shown close-to perfect weak scaling of HACC and CRK-HACC on all machines as reported in previous ECP milestone reports and reviews, justifying the extrapolation. A major achievement was the improvement of CRK-HACC performance leading to the excellent FOM measurement in 2020.

Machine	Code	n_p^3	t[s]	V[(h^{-1}Mpc)^3]	Nodes	FOM_{time}[s^{-1}]	FOM
Theta (Baseline)	CRK-HACC	2304^3	151.05	800^3	3072	54.62×10^7	
Theta (Baseline)	HACC	2304^3	13.57	800^3	3072		
Summit (2019)	CRK-HACC	7296^3	376.55	2523^3	4096	1295.56×10^7	23.72
Summit (2019)	HACC	7296^3	6.02	2523^3	4096		
Summit (2020)	CRK-HACC	7296^3	40.73	2523^3	4096	3946.06×10^7	72.25
Summit (2020)	HACC	7296^3	6.02	2523^3	4096		

large, high-resolution extreme-scale gravity-only simulation on Summit [30]. Currently, we do not see any obstacles for obtaining excellent performance on Frontier and Aurora to confirm the FOM measurement on the exascale machines.

Challenge Problems

The ExaSky scientific challenge problem will eventually comprise two very large cosmological simulations run with HACC on Frontier and Aurora that simultaneously address many science problems of interest. Setting up the challenge problem requires multiple simulations, now ongoing, which will be completed before the arrival of the exascale systems. These involve building subgrid models by matching against results from very high-resolution galaxy formation astrophysics codes[4] via a nested-box simulation approach and a medium-scale set for parameter exploration. The final two large-scale challenge problem runs on the exascale platforms will be based on these results.

The challenge problem runs are of two different types. The first is a large-volume, high mass and force resolution gravity-only simulation (specified in Table 51), and the second is a corresponding hydrodynamic simulation that includes detailed subgrid modeling (specified in Table 52). The main probes targeted with these simulations are strong and weak gravitational lensing shear measurements, galaxy clustering, clusters of galaxies, and cross-correlations that are internal to this set and with cosmic microwave background (CMB) probes, such as CMB lensing and thermal and kinematic Sunyaev-Zel'dovich (tSZ and kSZ) effect observations. The challenge problem runs will have the same cosmology and simulation volume.

Verification Simulations

In this section we describe the threshold simulation runs to demonstrate ExaSky readiness. The FOM ratio runs with HACC are designed to be full-machine (or as close to that as possible) but run with relatively few time-steps (5). The problem size and time to solution are used to determine the FOM ratio with respect to the baseline value.

Aside from meeting the FOM threshold, a requirement for ExaSky is to run a demonstration problem to verify that Challenge Problem simulations can be run successfully on Aurora and Frontier. This will be a test not just of the solvers and subgrid models but also of the full-scale analysis suite within HACC's CosmoTools library. The demonstration calculation will consist of three sets of simulations:

- An end-to-end gravity-only simulation fully equipped with all analysis tools at the same mass and force resolution as specified in challenge problem 1.

[4]in collaboration with members of the FIRE team; https://fire.northwestern.edu/about-fire/

Table 51: ExaSky challenge problem 1: Large-scale gravity-only simulation

Functional requirement	Minimum criteria
Initial conditions	Multiparticle Gaussian random field initial conditions using a specified linear power spectrum at the initial redshift based on ExaSky's distributed 3D FFT (SWFFT).
Boundary conditions	Periodic (box size of $3\,\mathrm{Gpc\,h^{-1}}$)
Force resolution	$\sim 1\,\mathrm{kpc}$
Mass resolution	$\sim 2 \times 10^8\,\mathrm{M_\odot}$, corresponding to $23{,}040^3$ simulation particles (final numbers are subject to adjustment due to system memory uncertainties).
Physics	Cosmological structure formation driven by the gravitational instability in an expanding universe (Vlasov-Poisson system of equations).
Numerical approach	N-body gravity via spectral particle-mesh (long-range) and direct particle-particle or tree/FMM (short-range). In situ analysis code integrated via HACC's CosmoTools library.
Science outputs	Summary statistics for matter and velocity fields, weak lensing shear, halo properties and halo spatial statistics, halo merger trees, and tSZ and kSZ statistics (post-processing). Light-cone outputs, galaxy summary statistics, strong/weak lensing for galaxy clusters, and sky maps for optical and CMB observables. Multi-probe cross-correlations.

Table 52: ExaSky challenge problem 2: Multi-scale cosmological hydrodynamics simulation.

Functional requirement	Minimum criteria
Initial conditions	Multiparticle Gaussian random field initial conditions using specified linear power spectra at the initial redshift based on ExaSky's distributed 3D FFT (SWFFT).
Boundary conditions	Periodic (box size of $3\,\mathrm{Gpc\,h^{-1}}$)
Force resolution	$\sim 1\,\mathrm{kpc}$
Mass resolution	$\sim 10^9\,\mathrm{M_\odot}$ (dark matter), $\sim 2 \times 10^8\,\mathrm{M_\odot}$ (baryons), corresponding to $2 \times 12{,}288^3$ simulation particles (final numbers will depend on system memory available).
Physics	Multi-scale cosmic structure formation and associated astrophysical modeling: gravitational evolution, gas dynamics, and subgrid models for astrophysical processes, including several feedback mechanisms.
Numerical approach	N-body gravity via spectral particle-mesh (long-range) and direct particle-particle or tree/FMM (short-range). Lagrangian hydrodynamics with CRK-SPH. Subgrid models for UV cooling and heating; star, black hole, and galaxy formation and associated effects; supernova and AGN feedback. (Subgrid models verified by cross-checks with Nyx.) In situ analysis code integrated via HACC's CosmoTools library.
Science outputs	Summary statistics for matter and velocity fields, Lyman-α forest, weak lensing shear, halo properties and halo spatial statistics, halo merger trees, and tSZ and kSZ statistics. Light-cone outputs, galaxy summary statistics, strong/weak lensing for galaxy clusters, and sky maps for optical and CMB observables. Multi-probe cross-correlations.

- Limited number of time steps with a fully representative simulation for the solvers (see Table 51), but not for the subgrid models (adiabatic run), demonstrating the ability to run at low redshifts (clustered regime) with predicted performance. Not turning on the subgrid models allows running with larger time-steps, making the test feasible with a smaller allocation.

- Smaller-scale simulation with subgrid models fully implemented, in this case run with a larger number of time-steps, again demonstrating the ability to function predictably at low redshifts. Scaling is not an issue when running subgrid models, therefore smaller volume simulations are adequate for testing purposes.

These tests will be designed to fit within the parameters of the ECP allocations available for ExaSky challenge problem tests; these are expected to be relatively small and not large enough for actually running the challenge problems. The size of the limited allocation depends on the results of the weak scaling tests, which are determined primarily by the performance of SWFFT, HACC's distributed 3D fast Fourier transform (FFT); this has historically never been an issue. The resource allocation on each system should correspond roughly to a few days of full machine runs, assuming that the systems are sufficiently stable.

Simulation Parameters

The simulation parameters describing the runs fall into two classes 1) specification of the cosmological and physical parameters, and 2) choices of particle number, box size, force and mass resolution, etc. the simulation parameters. The cosmological parameters will be fixed to the cosmology used for the baseline run (WMAP-7) and other parameter choices are made either to 1) yield results consistent with current observations or 2) to be consistent with known physical models and terrestrial measurements. Lastly, the hydro simulations include additional specifications for subgrid models. Our challenge problem science goals require the inclusion of radiative (metal-line) cooling, star formation and chemical enrichment, galactic winds, and AGN feedback prescriptions, which have all been incorporated into our solvers. Each model includes a number of parameters that are selected based on calibration runs targeting the relevant observables (e.g., Lyman alpha flux, galaxy stellar mass functions, SZ cluster measurements, etc.).

The simulation parameter inputs are relatively straightforward as we do not have a complex "input deck" (domain geometry, databases, etc.). The input physics and simulation parameters are specified in short input files. We have considerable experience running on multiple generations of new HPC platforms and have several times been among the very first users of the machines. The HACC build-system is deliberately kept very simple to minimize software dependencies. We have makefiles tailored to the different architectures to allow for different compilation, module, and code settings. Runtime settings can very quickly be changed to adjust to, e.g., the stability of the machine to maximize utilization. Table 53 provides a summary of the problem size for both Challenge Problems (Challenge Problem 1 will be carried out with HACC, Challenge Problem 2 with CRK-HACC). We provide information about recent simulations on Spock and then—assuming that 20% of the exascale machines will be available for the threshold simulations—provide estimates of the size of these runs (scaling these runs to smaller/larger node counts is straightforward). The exact specifications will depend on the final memory available on the systems.

Simulation Artifacts

HACC outputs are classified into three levels: 1) Level 1 outputs corresponding to raw particle data, 2) Level 2 outputs, corresponding to partial in situ analysis, which produces reduced outputs for post-processing, and 3) Level 3 outputs, which are science-ready data sets produced either in situ or in post-processing. A number of these outputs are used to verify the correctness of the obtained results.

More specifically, the code outputs include raw and downsampled particle outputs, density power spectra (for every particle species), VTK files for visualization and health monitoring purposes at every time step, halo finder outputs that allow for building halo merger trees in post-processing, "core" particle tracking outputs (a method for tracking halo substructure), particle lightcone outputs, galaxy catalogs (including stellar, AGN, and gas masses, metallicity, etc.), Lyman-α power spectra, global subgrid statistics (e.g., total star formation rate), and density-temperature phase diagrams.

For the gravity-only simulation, we will immediately provide measurements for the density power spectra (Fig. 37) at regular intervals and will compare the results to previous simulations [30, 31]. These measurements have proven in the past to be an excellent indicator for possible problems with the machine or the simulation.

Table 53: ExaSky challenge problem specification and downscaled runs. The simulations on Spock have been run already and we report on these in § 5.2.2. We assume that 20% of the machine is available for the simulations. We also assume that the final machine (Frontier in this case) has a node count of approximately 10,000 with 4 GPUs/node. Run sizes can be adjusted according to machine memory and system availability.

Simulation	Machine	Code	n_p^3	$V[(h^{-1}Mpc)^3]$	Nodes
Early test	Spock	HACC	2144^3	286^3	16
Early test	Spock	CRK-HACC	600^3	150^3	4
Downscaled run	Exascale hardware	HACC	13312^3	1762^3	2048
Downscaled run	Exascale hardware	CRK-HACC	7168^3	1622^3	2048
Challenge Problem	Exascale hardware	HACC	23048^3	3000^3	~10000
Challenge Problem	Exascale hardware	CRK-HACC	12288^3	3000^3	~10000

While the run is progressing, we will generate visualizations from specific ranks. HACC provides information about the slowest and fastest ranks throughout the simulation and the visualization output can be adjusted to capture these ranks as well as any other chosen ranks. These visualizations will also provide an immediate check for the health of the simulation. Starting at redshift, $z \sim 3$, we will measure the halo mass function and mass-concentration relation. As for the power spectrum mentioned above, we can compare the results to previous simulations. Finally, we will closely monitor the wall-clock times for the solver and analysis tools to identify any possibly unexpected bottlenecks. These artifacts will allow us to ensure that the code is running correctly and the performance is as expected to enable the actual Challenge Problem run. At a later stage, more post-processing analysis will be carried out (building of merger trees, halo lightcones, density maps etc.). While the machinery for generating these artifacts is available within HACC's CosmoTools library, it will take some more time and computational resources to carry out these tasks.

The hydro simulations include the same verification artifacts as the gravity-only runs, where the measurements are taken for each individual species. These include stellar, baryon and dark matter power spectra, as well as representative mass functions for each species. We also provide hydro-specific artifacts, including measurements of state variable (density, temperature, entropy) phase diagrams, and Lyman-α power spectra that can be compared against our validation runs with the Nyx code (Figure 38) as well as the relevant calibrated observables (e.g. galaxy stellar mass function, global star formation rate, etc.). Lastly, in addition to monitoring the wall-clock timings, our hydro simulations provide baryon specific state-of-health outputs including conservation checks—such as mass conservation from chemical enrichment—and adaptive time stepping outputs to verify the simulation is running smoothly and the particle properties are physical.

Meeting Minimum Requirements

Prior to the threshold runs, we will carry out smaller verification simulations to test out all of the analysis codes on the system. Comparison of results against these outputs, observational validation data sets, and a number of convergence tests that have been produced in earlier simulations will be used to verify that the minimal accuracy and performance criteria have been met. At this point we will be ready for the threshold simulations.

For ExaSky, meeting the minimum requirements consists of two parts:

- Running an upscaled version of the baseline FOM ratio problems (gravity-only and adiabatic hydro) originally carried out on Theta, and verifying that the FOM ratio exceeds 50.
- The executed downscaled challenge problem(s) described above must meet the following conditions to be considered successful: 1) Expected performance as predicted from previous scaling runs for the solvers and the analysis code suite; 2) All CosmoTools analysis codes to run successfully in both in situ and post-processing modes and produce the expected results, based on convergence expectations, verification/validation tests, and the smaller-scale runs.

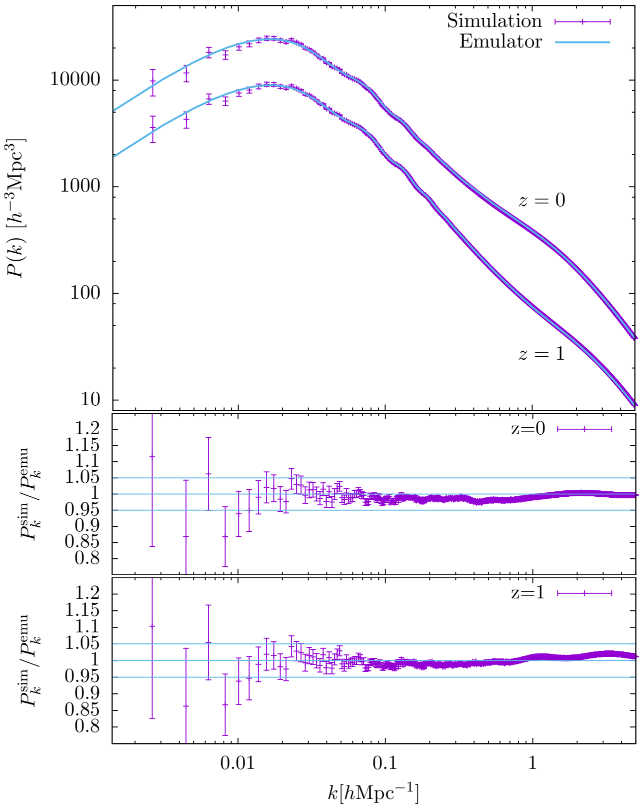


Figure 37: Example of power spectrum outputs from HACC cosmological simulations. Upper panel: Power spectrum results at redshift $z = 0$ and $z = 1$ from the "Last Journey" simulation [31]. Solid lines show predictions from the Cosmic Emulator [32]. Lower panels: Ratio of the simulation and the emulator at $z = 0$ and $z = 1$. The light blue bands show a 5% range. The results are within the emulator accuracy bounds reported in Ref. [32].

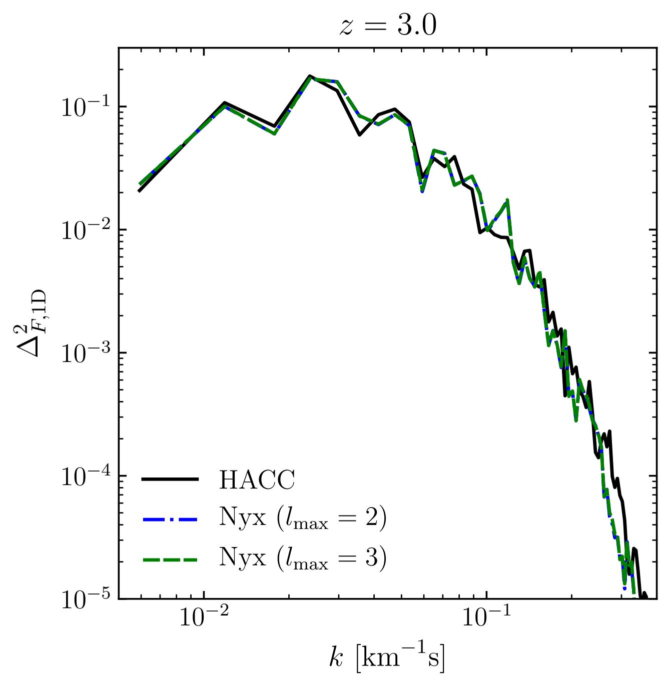


Figure 38: Example comparison of HACC and Nyx (two different AMR levels) one-dimensional Ly-alpha flux power spectra with the y-axis in dimensionless units and the x-axis scaled to velocity units.

5.2.2 Early Hardware Status

The work on early hardware systems has been reported in a number of ExaSky milestone reports, most specifically in the report on Milestone ADSE01-45, covering the successful code tests on the systems Iris and Yarrow (JLSE) and Tulip (OLCF) using a variety of programming models (OpenCL, CUDA, HIP, SYCL/DPC++). This milestone also included work on CosmoTools. Here we also describe work on Spock (OLCF).

Spock Gravity-Only Simulations

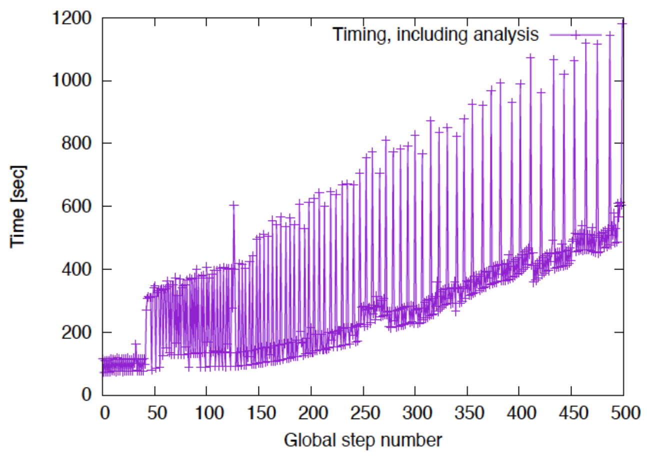
We have carried out a range of tests and simulations on Spock to verify the expected performance of the code and is scalability. The most important experiment was a run on 16 nodes on Spock at the same resolution as for the challenge problem. The run was restricted to 16 nodes to avoid blocking a large portion of the machine for other users; we carried out faster 32 node runs at lower resolution as well. The overall settings for the simulation (time stepping, initial conditions, in situ analysis settings) have been used in major extreme-scale runs on Summit and Mira in the past [30, 31] and have been carefully verified.

In order to mimic the Challenge Problem for the gravity-only run, we specified the following simulation parameters:

$$n_p^3 = 2144^3\,, \quad L_{\rm box} = 286\,{\rm h}^{-1}\,{\rm Mpc}\,, \tag{7}$$

and a force resolution of $\sim 1\,{\rm h}^{-1}\,{\rm kpc}$. This leads to a particle mass of $m_p = 2 \times 10^8\,{\rm h}^{-1}\,{\rm M}_\odot$. The simulation used 16 nodes and the available 16 GPUs. This set-up is roughly a factor of 1200 smaller than the Challenge Problem in volume. We anticipate that the Frontier nodes will have 2× more memory on the CPU host and close to 10,000 nodes. As shown in previous reviews, HACC weak-scales essentially perfectly on Summit and we anticipate the same weak scaling performance on Frontier. Therefore, this downscaled run should be very informative for the timing we can expect on Frontier (note that we expect to do even better with the MI200 GPUs compared to the MI100s in Spock).

The simulation was fully equipped with all the in situ analysis steps we will carry out in the Challenge Problem simulation. This includes regular evaluation of the power spectrum, halo finding, core tracking



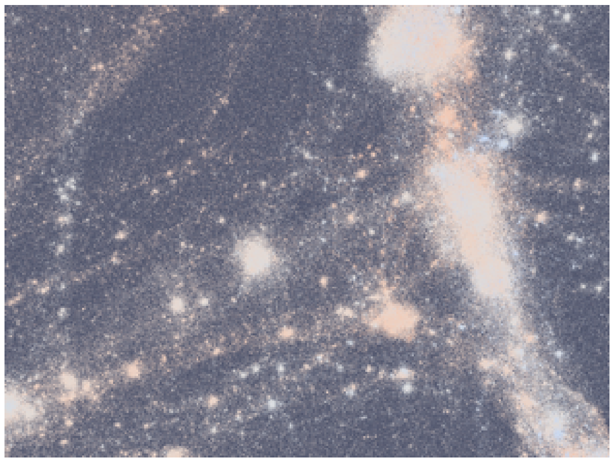


Figure 39: Left: Timing information from the downscaled Spock run—each global time step includes a number of sub-cycles. The resolution is the same as for the Challenge Problem and the work load per node is half of what we would use on Frontier, taking into account Spock's limited memory footprint. Close to Step 50, the in situ analysis tasks switch on, as can be seen by the regular increase in run-time. Around Step 250, the particle lightcone outputs start, again, marked by a small increase in timing. The overall slow down is due to the formation of nonlinear structure. Given the high mass resolution of this simulation, the slow-down by 4× is small and the performance of our load-balancing scheme is excellent. We include all operations carried out during the run, e.g., IO, restarts, etc., providing a very realistic view of what can be expected for the challenge run. Right: An image generated from our in situ visualization outputs at the final time step—a zoom-in showing particles colored by velocities.

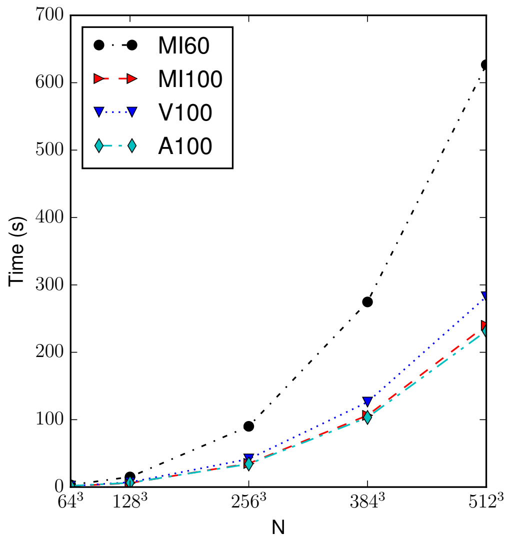
for substructure investigations, and particle lightcone outputs. In addition, we also stored downsampled particle files, a handful of full particle files, particle information for visualization purposes, and we did check-point/restarts at regular intervals. The left panel in Fig. 39 shows a summary of the results from this run with regard to timing. Overall, the simulation took just under 46 hours to complete, including all analysis tasks and IO.

Hydro Simulation Performance on Early Hardware

We have devoted a number of milestone reports to investigating the performance of HACC hydro kernels on early hardware. The left panel of Fig. 40, demonstrates an aggregate timings comparison of a benchmark run on AMD (MI60 and MI100) and NVIDIA (V100, A100) devices. In this test, we measured the aggregate timings of the first 5 time-steps of an adiabatic cosmology simulation with $N = 64^3$, 128^3, 256^3, 384^3 and 512^3 particles in volumes of sizes $L = 50.3975$, 100.795, 201.59, 302.385 and 403.18 h^{-1} Mpc. The A100 and MI100 have similar hardware specifications, and demonstrate matching performance in our test runs. Our results indicate that we are achieving the same performance per Flops between the two different vendors, indicating that our porting from CUDA to HIP was successful.

The right panel of Fig. 40 shows the simulation timings of the downscaled challenge problem listed in Table 53, a simulation of $N = 600^3$ particles in a box of side length $L = 150\,\mathrm{h}^{-1}$ Mpc on 4 nodes. This simulation includes star formation and wind feedback subgrid models run all the way to redshift $z = 0$. The run served as a useful mock simulation of the anticipated downscale run on Frontier, and facilitated testing of both the subgrid solvers, as well as the analysis pipelines. As we refine and finalize our subgrid kernels and output products, we will perform similar end-to-end simulation runs on both Spock and Crusher (when it becomes available).

5.3 EQSIM



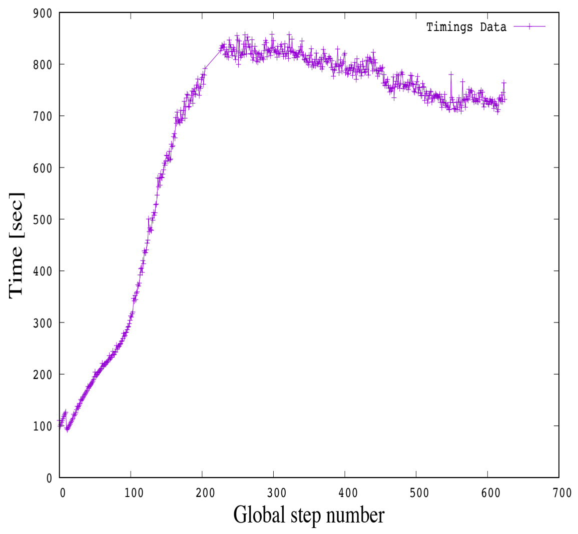


Figure 40: (Left): Aggregate timings of 5 timesteps in a suite of benchmark adiabatic hydro simulations used to compare kernel performance between AMD (MI60, MI100) and NVIDIA (V100, A100) GPUs. We note the matching timings between the A100 and MI100 benchmarks, encouraging given their similar hardware Flops specifications. Our results indicate that the porting from CUDA to HIP was successful and the AMD hardware is performing as expected. (Right): The timings for the downscaled challenge problem simulation listed in Table 53. This run includes star formation and galactic wind subgrid models, which tested a full simulation to redshift $z = 0$ on early hardware.

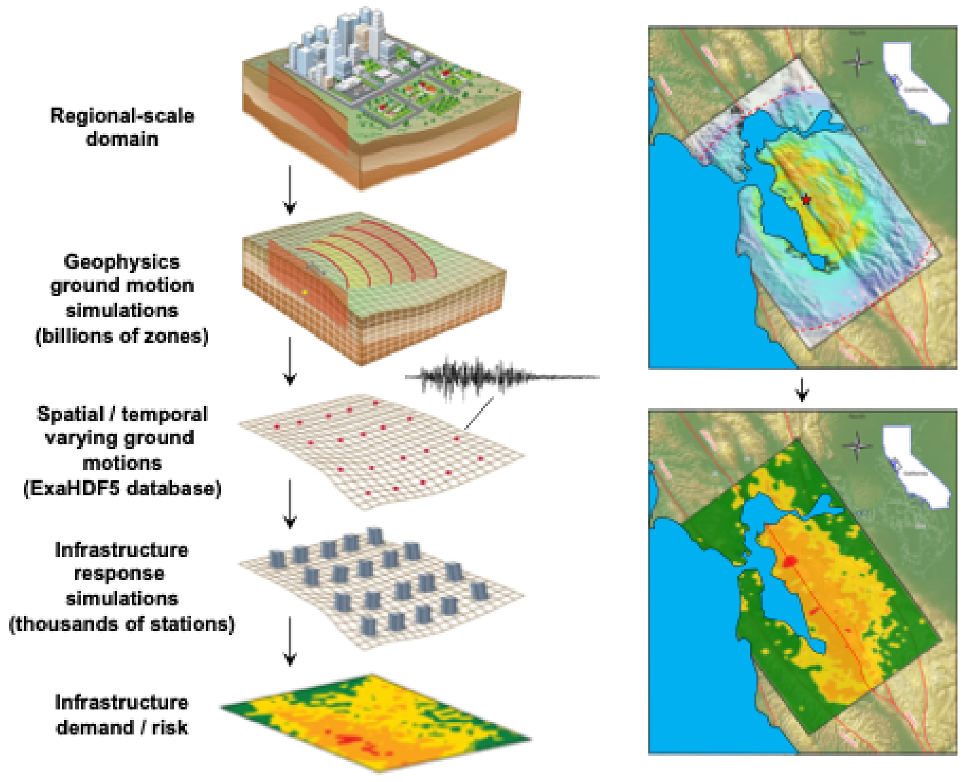

Regional-scale domain
Geophysics ground motion simulations (billions of zones)
Spatial / temporal varying ground motions (ExaHDF5 database)
Infrastructure response simulations (thousands of stations)
Infrastructure demand / risk

Figure 41: EQSIM end-to-end (fault-to-structure) earthquake simulations.

5.3.1 *KPP Verification Plan*

Traditional earthquake hazard and risk assessments for critical facilities have relied on empirically based approaches that use historical earthquake ground motions from many different locations to estimate future earthquake ground motions at a specific site of interest. Because ground motions for a particular site are strongly influenced by the physics of the specific earthquake processes, including the fault rupture mechanics, seismic wave propagation through a heterogeneous medium, and site response at the location of a particular facility, earthquake ground motions are very complex with significant spatial variation in frequency content and amplitude. The homogenization of many disparate records in traditional empirically based ground motion estimates cannot fully capture the complex site-specificity of ground motion. Over the last decade, interest in using advanced simulations to characterize earthquake ground motions (earthquake hazard) and infrastructure response (earthquake risk) has accelerated significantly. However, the extreme computational demands required to execute hazard and engineering risk simulations at regional scale have been prohibitive. One fundamental objective of the EQSIM AD project is to advance regional-scale ground motion simulation capabilities from the historical computationally limited frequency range of 0–2 Hz to the frequency range of interest for a breadth of engineered infrastructure of 0–10 Hz. Another fundamental objective of this project is to implement an HPC framework and workflow that directly couples earthquake hazard and risk assessments through an end-to-end simulation framework that extends from earthquake rupture to structural response, thereby capturing the complexities of interaction between incident seismic waves and infrastructure systems (Fig. 41).

To achieve the overall goals, two fundamental challenges must be addressed. First is the ability to effectively execute regional-scale forward ground motion simulations at unprecedented frequency resolution with much larger, much faster models. Achieving fast earthquake simulation times is essential for allowing the necessary parametric variations needed to span critical problem parameters (e.g., multiple fault rupture scenarios). Second, as the ability to compute at higher frequencies progresses, there will be a need for better characterization of subsurface geologic structures at increasingly fine scales; thus, a companion schema for representing fine-scale geologic heterogeneities in massive computational models must be developed. To

Table 54: EQSIM challenge problem details

Functional requirement	Minimum criteria
Physical phenomena and associated models	Earthquake simulations, including representative fault rupture mechanics, wave propagation through heterogenous 3D geologic structure, and appropriate coupling between regional geophysics and local soil/structure models to fully represent infrastructure response to complex incident seismic waveforms.
Numerical approach and associated models	Geophysics simulations will be executed with a fourth-order, summation-by-parts finite difference program (SW4) that will require extensive advancement to achieve ground motion simulation goals. Achievement of Exascale challenge goals will require advanced algorithms, code optimization on GPU accelerator platforms and exascale platform hardware. Infrastructure simulations will be based on appropriate rigorous coupling of regional-scale geophysics simulations with local soil-structure models.
Simulation details	Regional-scale simulations will typically encompass a large urban region surrounding the urban environments of interest and the regional earthquake faults (sources) of interest. A representative model for the SFBA was developed with a finite difference domain, including on the order of 200–300 billion grid points for high-frequency resolution simulations.
Demonstration calculation requirements	The EQSIM science demonstration runs, which are requirements performed annually in the project milestone plan to establish current application FOM, will revolve around the SFBA model with a simulation of a representative $M = 7$ earthquake on the Hayward fault and a simulation of a corresponding 90–120 seconds of earthquake motions. These runs consistently measure the project's annual progress toward the exascale challenge problem and increases in the EQSIM FOM.

evaluate regional-scale simulations and assess progress on the application developments of this project, a representative large regional-scale model of the San Francisco Bay Area (SFBA) has been created (Fig. 42). This model includes all the necessary geophysics modeling features, such as 3D geology, earth surface topography, material attenuation, nonreflecting boundaries, and fault rupture models. For a 10 Hz simulation, the computational domain includes approximately 200–300 billion grid points in the finite difference domain depending on the degree to which soft, near-surface sediments are resolved in the computational model. The SFBA model provides the basis for testing and evaluating advanced physics algorithms and computational implementations, and the functional requirements of this computational challenge are summarized in Table 54.

Verification Simulations

Facility Requirements

Essential software requirements include: MPI, RAJA (an ECP supported library for efficient application implementation on GPU-based systems), HDF5 (ECP supported I/O libraries) and ZFP (ECP supported data compression). No additional third-party libraries are required.

Input Description and Execution

The demonstration problem input will consist of the existing SW4 input model for the San Francisco Bay Area. The existing baseline model includes a large 120km x 80km x 30km domain surrounding the urban regions of the San Francisco Bay Area (Fig. 42). This model is based on the United States Geological

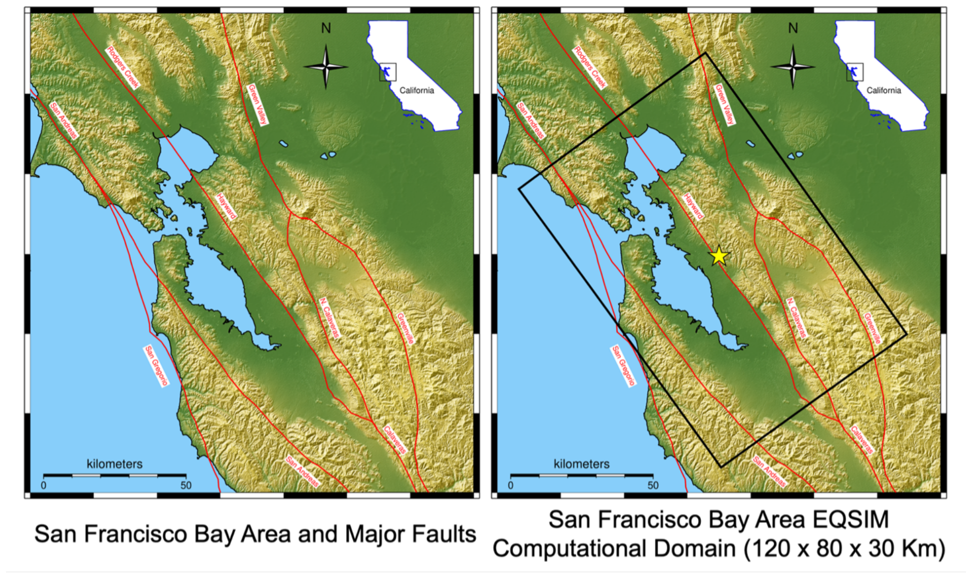


Figure 42: The EQSIM San Francisco Bay Area (SFBA) regional scale model.

Survey three-dimensional geologic dataset for the SFBA and includes surface topography (which increases the computational demands), a semi-stochastic detailed Hayward fault rupture model, geologic material attenuation, and non-reflecting boundaries at the edges of the computational domain. The SW4 computational grid consists of a combined near-surface curvilinear grid to represent surface topography and an efficient Cartesian grid at depth. Both the curvilinear and Cartesian grids have mesh refinement capabilities to allow optimal matching of the grid to typical depth-dependent geologic properties. The discretization and resulting number of grid points in the execution model depends on:

- The frequencies that are to be resolved in the ground motion simulations, per the EQSIM challenge problem goal and the desire to resolve frequencies relevant to engineered systems, the ground motions will be resolved to 10 Hz in the challenge problem demonstration;
- The minimum shear wave velocity resolved in soft near-surface sediments, historical regional-scale simulations have typically utilized a cut-off shear wave speed ($V_{s\,\mathrm{min}}$) on the order of $500\,\mathrm{m\,s^{-1}}$, however, the demonstration problem will attempt to push to even lower near-surface shear wave speeds to better capture the influence of soft sediments on earthquake shaking. The demonstration calculation will attempt to push $V_{s\,\mathrm{min}}$ to the range of $250\,\mathrm{m\,s^{-1}}$ or even $150\,\mathrm{m\,s^{-1}}$ depending on realized performance on Exascale platforms. This will allow representation of ground motion amplifications in areas where there are very soft near-surface sediments and establish new computational benchmarks.

As described in detail in the section below, the verification of KPP-1 performance achievement will be demonstrated by working through a progressively more challenging set of regional earthquake simulations. Based on progress to-date on Summit, and potential Frontier performance, the resource requirements shown in Table 55 are the best estimates.

Table 55 refers to the resource requirements to complete full runs of the challenge problems, which the EQSIM team certainly wants to demonstrate. However, it should be noted that due to the nature of the EQSIM ground motion simulations, accurate projections of KPP demonstration runs can be reliably executed with more modest compute resources. The computational effort per time step remains quite constant throughout an earthquake simulation so running a given large problem for a modest number of time steps with the creation of one representative checkpoint file, can provide a very accurate extrapolation of the requirements, and associated performance, for a very large KPP run.

Simulation Artifacts

The principal data created from the large SFBA simulations includes three components of earthquake

Table 55: EQSIM best estimate computational resources for KPP-1 verification sequence.

Simulation	Node-hours (full run)	Memory	Disk
M7 Hayward fault earthquake 10 Hz resolution $V_{s\,\mathrm{min}} = 500\,\mathrm{m\,s^{-1}}$	Nodes = 392, Hours = 5.2, Node-hours = 2038	45 TB	26 TB
M7 Hayward fault earthquake 10 Hz resolution $V_{s\,\mathrm{min}} = 250\,\mathrm{m\,s^{-1}}$	Nodes = 700, Hours = 10.5, Node-hours = 7350	81 TB	46 TB
M7 Hayward fault earthquake 10 Hz resolution $V_{s\,\mathrm{min}} = 150\,\mathrm{m\,s^{-1}}$	Nodes = 2700, Hours = 18, Node-hours = 48,600	316 TB	150 TB
Stretch Goal M7.5 San Andreas fault 10 Hz resolution $V_{s\,\mathrm{min}} = 500\text{–}250\,\mathrm{m\,s^{-1}}$	A stretch goal on Frontier will be to execute a much larger M7.5 earthquake on the San Andreas fault to 10 Hz resolution which is expected to require up to $\sim$ 6300 Frontier nodes.		

ground motion time histories at the grid points throughout the computational domain. Currently in the EQSIM framework, ground surface velocity time histories are stored across the entire domain to allow subsequent infrastructure response simulations based on fixed-base models (the EQSIM weak coupling case). In addition, grid point motions from a 3D volume near the surface of the model can be compressed and stored to allow rigorous coupling between the geophysics model and local engineering models of soil-structure systems. The creation of a full dataset of ground motions is the principal output of the demonstration calculation and successful creation of the output files and correctness of solution will be confirmed by comparison with previously well documented executed runs for the SFBA (Fig. 43) and usage of the ground motion data in structural system earthquake risk assessments. In addition, the parameters of the demonstration calculation necessary to compute the EQSIM FOM, including frequency resolved, minimum shear wave velocity resolved and wall clock time for a M7 Hayward fault earthquake, will all be documented and rolled into a FOM calculation.

KPP Verification

The specific FOM which has been developed for EQSIM is

$$\mathrm{FOM} = \frac{(\mathrm{Freq})^4}{\text{wall clock time} \times 7.6} \left(\frac{500}{V_{s\,\mathrm{min}}}\right)^4 , \tag{8}$$

where

Freq:	the highest frequency (Hz) resolved in the regional ground motion simulation;
wall clock time:	wall clock time (hours) of one full rupture scenario simulation for a large M7 earthquake on the Hayward fault (typically simulating on the order of 90 seconds of physical earthquake rupture and subsequent wave propagation);
$V_{s\,\mathrm{min}}$:	lowest shear wave speed represented in the computational model (m/s);
7.6:	normalization factor so that the application is baselined to a FOM of 1.0 in our first regional scale simulations at the start of the project, with $V_{s\,\mathrm{min}} = 500\,\mathrm{m\,s^{-1}}$ in the regional model.

The fourth power on the minimum shear wave velocity ($V_{s\,\mathrm{min}}$) and frequency terms represent the fact that the smallest wave length depends on their ratio. For time integration in 3D this results in a computational effort that varies as the frequency over $V_{s\,\mathrm{min}}$ ratio to the power of four.

The progress in EQSIM to-date in terms of peak FOM achieved is summarized in Table 56. The KPP-1 verification runs will be executed on Frontier, and Aurora as available, as soon as access is available.

The work-up to the KPP-1 verification runs on Frontier (and Aurora) will consist of a sequence of full Hayward fault earthquake simulation runs with progressively lower shear wave velocities to account for the effect of soft near-surface soils (Table 56). The plan is to execute a full simulation of a M7 Hayward fault

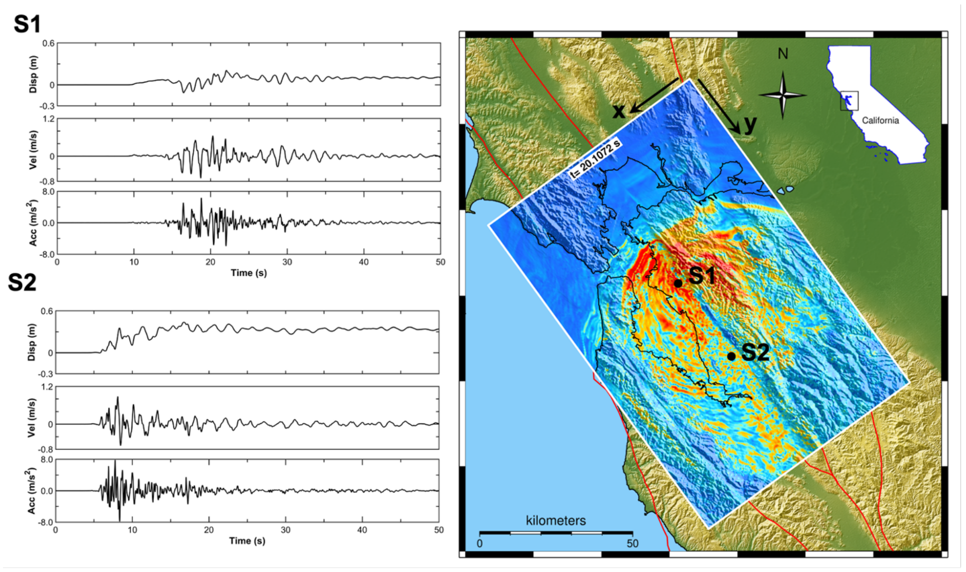


Figure 43: Artifacts for evaluation of the KPP demonstration calculations—well established SFBA simulated ground motions from multiple earthquake simulations performed throughout the life of the EQSIM project.

Table 56: EQSIM performance increases and KPP-1 verification runs.

Benchmark simulation (Platform)	Code attributes	$V_{s\,\mathrm{min}}$	Frequency resolution (Hz)	Compute nodes	Wall clock time (h)	FOM
1 (Cori)	Initial run of SW4 ported to Cori	$500\,\mathrm{m\,s^{-1}}$	3.67	2048/9668	23.9	1.0
2 (Cori)	SW4 with optimized hybrid MPI/OpenMP loops	$500\,\mathrm{m\,s^{-1}}$	4.17	6528/9668	12.0	3.32
3 (Cori)	SW4 with Cartesian mesh refinement	$500\,\mathrm{m\,s^{-1}}$	4.17	4000/9668	6.0	6.63
D (Cori)	SW4 using all of the Cori computer	$500\,\mathrm{m\,s^{-1}}$	5.0	8192/9668	9.2	8.95
E (Summit)	Initial run of SW4 ported to the Summit computer	$500\,\mathrm{m\,s^{-1}}$	10.0	1200/4600	19.9	66.2
F (Summit)	Fall 2020 run of SW4 including enhanced I/O, curvilinear and Cartesian mesh refinement	$500\,\mathrm{m\,s^{-1}}$	10.0	1024/4600	6.9	189
G (Summit)	Fall 2021 run of SW4 including enhanced mesh refinement	$250\,\mathrm{m\,s^{-1}}$	10.0	922/4600	23.7	888
8 (Frontier)	KPP-1 verification runs	$500–150\,\mathrm{m\,s^{-1}}$	10.0	TBD	TBD	TBD
9 (Aurora)	KPP-1 verification runs	$500–150\,\mathrm{m\,s^{-1}}$	10.0	TBD	TBD	TBD

event first with $V_{s\,\min}$ of $500\,\mathrm{m\,s^{-1}}$, then with $V_{s\,\min}$ of $250\,\mathrm{m\,s^{-1}}$ and finally with $V_{s\,\min}$ of $150\,\mathrm{m\,s^{-1}}$. This progression of increasingly computationally demanding problems will achieve progressively higher application FOM values and will result in a very large FOM increase relative to the performance at the inception of the EQSIM development.

5.3.2 Early Hardware Status

The EQSIM SW4 geophysics code for seismic wave propagation has been ported to both Spock (pre-Frontier hardware) and Arcticus (pre-Aurora hardware). The full SW4 application has been successfully compiled on both Frontier and Arcticus and the entire suite of ~ 40 SW4 regression test problems have been successfully completed on both platforms with verification of correct answers. However, as discussed subsequently, a number of significant performance-degrading issues remain. Given the critical path of Frontier as the first arriving exascale platform, the EQSIM team is most concerned about significant performance-degrading issues identified during early testing on Spock.

5.4 Subsurface

5.4.1 KPP Verification Plan

Verification Simulations

The Subsurface validation problem is defined in Walsh, Mason, Frane, and Carroll [33]. GEOSX is run on a $3.5\,\mathrm{cm} \times 1.5\,\mathrm{cm} \times 0.25\,\mathrm{cm}$ domain with $100\,\mathrm{\mu m}$ resolution. The Chombo-Crunch part of the simulation covers an initial fracture with dimensions $3.5\,\mathrm{cm} \times 0.008\,\mathrm{cm} \times 0.25\,\mathrm{cm}$ and requires $3072 \times 32 \times 256$ cells, $\sim 10\,\mathrm{\mu m}$ mesh spacing, and 768 32^3 boxes. On Summit, the problem layout requires 1–3 nodes for GEOSX and 4 nodes for Chombo-Crunch, where the problem is decomposed into 192 boxes per node.

Using these parameters we can extrapolate to estimate the resource allocation required for larger problems. For example,

$$4\,\text{nodes} \times 6\,\text{GPUS/node} \times 1\,\text{MPI rank/GPU} \times 7\,\text{threads/MPI rank} = 168\,\text{threads}\,,$$

Using the validation problem of 768 boxes, this yields 4.5 boxes/thread or 32 boxes/MPI rank. The simulation time on Summit for 9 d of simulated time for experimental validation of Walsh, Mason, Frane, and Carroll [33] for fracture evolution requires 10 h of wall clock time to run approximately 2000 flow and transport substeps and 100 hydro-mechanical-reactive steps.

The parameters and resource allocation estimates for the baseline, intermediate, and full challenge problem runs are listed in Table 57. We will perform a baseline challenge problem that makes use of the lower end of Frontier exascale resources by increasing the resolution of the validation test problem by a factor of 8 to approach 1 micron resolution covering the open fracture from 0.01–0.1 cm. The intermediate challenge problem extends the baseline challenge problem to 10 cm length for capture of fracture evolution. The full challenge problem extends the intermediate problem to wider apertures during fracture evolution. The estimated runtimes for Chombo-Crunch are for 10 d of simulated time applied to Li, Steefel, and Jun [34] to reconcile the reaction time of CO_2 invasion in cement with approximately 4000 substeps and 200 hydro-mechanical-reactive steps. Each Chombo-Crunch simulation is estimated to require 40 hours of wall clock time.

The stretch problem for GOESX is wellbore integrity. We will perform a series of simulations, optimizing each physics component—mechanics/deformation, compositional multiphase flow in fracture/cement/rock, and reactive transport—and the coupled problem. The proposed problem size is $10 \times 1 \times 0.1\,\mathrm{m}^3$ yielding 100 M elements and 1 B DOF.

Simulation Artifacts

Output files include HDF5 plot files of the state variables and checkpoint files for restarts. These files are dumped after a physics-based event in the simulation (e.g., fracture deformation, dissolution due to reactions). We use experimental data to validate the time scales of fracture evolution as well as aperture opening lengths in the challenge problem. Plot files are post-processed with visualization software, VisIt, to obtain the validation data as demonstrated previously in milestone ADSE05-20 (hydraulic aperture from 1–9 d).

Table 57: Subsurface baseline, intermediate, and full challenge problem specifications and runtime estimates. The GEOSX domain size is the overall problem size, and the Chombo-Crunch domain size covers the pore scale window. All Chombo-Crunch runtimes are estimated for 40 hours.

Code	Parameters	Baseline	Intermediate	Full
GEOSX	domain (cm^3)	$3.5 \times 1.5 \times 1.5$	$10.5 \times 1.5 \times 1.5$	$10.5 \times 1.5 \times 1.5$
	resolution (µm)	100	100, 20	100, 20
	nodes	3	3, 375	3, 375
Chombo-Crunch	domain (cm^3)	$3.5 \times 0.008 \times 0.25$	$10.5 \times 0.008 \times 0.25$	$10.5 \times 0.012 \times 0.25$
	mesh size	$24{,}576 \times 256 \times 2048$	$73{,}728 \times 256 \times 2048$	$73{,}728 \times 384 \times 2048$
	boxes	393,216	1,179,648	1,769,472
	nodes	2048	6144	9216
	node-hours	81,920	245,760	368,640

Standard simulation output will be piped to a text file displaying the steps taken in the code and status of the code solvers throughout the simulation and their usage of processors. Chombo also has a built-in time analyzer that dumps a `time.table.#` for each MPI process. Additional files include `PETSc.history` to detail GPU usage for solvers in Chombo-Crunch—actual GPU flop rates in GPU MFlops, CPUtoGPU Counts and Sizes and GPUtoCPU Counts and Sizes that correlate with timestep output in the `pouts`. We also have instrumentation for time spent on device in other components of code like in CrunchPP reactions that shows up in `pouts` as time spent in CUDA kernels (`assemble_local`, `jacmin`, `os3dnewton_lu_solve`) in ms. GEOSX is built with Caliper and dumps Caliper output to measure performance in GEOSX, and in particular hypre-CUDA solves. Batch job output and error files will display the success of the simulation and errors encountered from the execution of the batch job script.

Verification of KPP-2 Threshold

The exascale challenge problem is an extension of the validation test problem (larger domain, higher resolution). The validation test problem produces data from the simulation that can be (and has been for smaller domains with coarser resolution) compared to experimental data for CO_2 invasion in cement. In particular, to meet the science goals of the challenge problem, we simulate approximately 9–10 d of time and compute hydraulic aperture as in Walsh, Mason, Frane, and Carroll [33]. Simulated 8–9 d is also required to capture the diffusive timescale for the reactive components to affect the porosity in the rock matrix which in turn changes mechanical properties.

For verification of the KPP by an SME, only 1 d of simulated time of the exascale challenge problem(s) would be necessary to demonstrate that the code is doing the right thing. This would allow for validation against 1 experimental data point and only require 10% of the allocation requested above. Alternatively or additionally, the smaller validation problem could be run on Frontier to also verify usage of the hardware without using a large portion of the machine. This would allow for validation of the capability to address the scientific goals of the project: *(i)* 9 d of simulated time to capture diffusive time scale in rock matrix and *(ii)* 9 d of data to validate with experimental hydraulic aperture.

5.4.2 Early Hardware Status

We have ported the Chombo-Crunch flow solver to Spock. We reported performance numbers very recently in milestone ADSE05-67 that showed faster runtimes on Spock than Summit and Tulip for the EBProto EBINS flow benchmark as shown in Table 58. We also reported roofline models on Spock and Perlmutter for the EBProto kernel (see Figs. 44 and 45).

Testing on Spock is limited due to the need for a stable PETSc installation with a HIP version of hypre. There are known bugs with Cray. We have been told that these should be resolved once ROCm 4.5.0 and November PE are available. GEOSX has started porting work on Spock. We have been waiting for better support from GEOSX dependencies.

We will port to Crusher as soon as it is available, which is projected for December 2021. But our experience

Table 58: EBProto EBINS flow benchmark runtimes in s on various platforms.

Cori (V100)	Summit (V100)	Tulip (MI60)	Tulip (MI100)	Spock (MI100)	DGX (A100)	Perlmutter (A100)
196.32	216.89	314.90	265.86	188.28	136.26	110.73

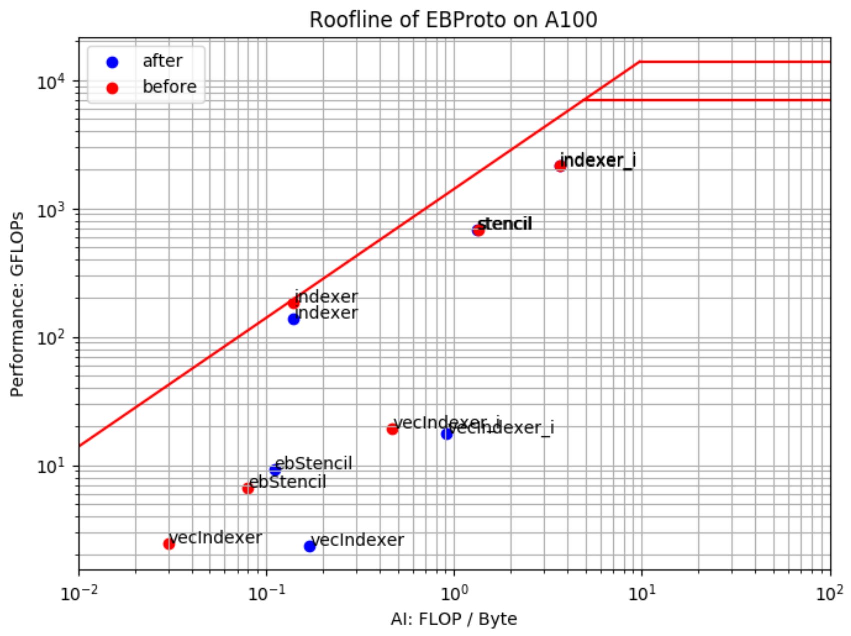


Figure 44: Roofline plot for Chombo-Crunch EBProto kernel on Spock (AMD MI100).

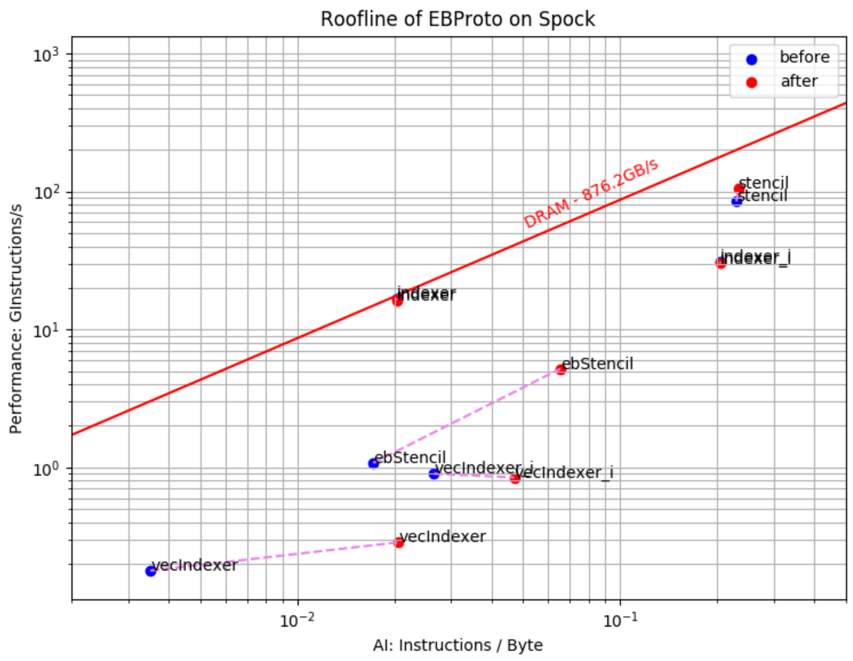


Figure 45: Roofline plot for Chombo-Crunch EBProto kernel on Perlmutter (NVIDIA A100).

Table 59: Subsurface packed spheres weak scaling test on Perlmutter and Summit.

Perlmutter time 1 thread	Perlmutter time 32 threads	Perlmutter MPI ranks (nodes)	Summit time 1 threads	Summit time 7 threads	Summit MPI ranks (nodes)
5.61	3.36	120 (30)	4.52	2.29	240 (40)
6.57	4.23	240 (60)	5.36	4.69	480 (80)
7.29	4.66	480 (120)	6.41	5.95	960 (160)
9.32	6.56	960 (240)	7.56	8.15	1920 (320)
10.43	7.75	1920 (480)	10.98	14.14	3840 (640)
10.73	8.00	3840 (960)	13.84	x	7680 (1280)

is that it will be a slow performance portability cycle to Crusher with the software stack and installation of dependencies PETSc and hypre. We are not porting to Arcticus as our challenge problem will be performed on Frontier, not Aurora. In lieu of advanced performance results on Spock, we present results from Perlmutter and Summit in Table 59 to demonstrate successful performance portability on a GPU platform.

5.5 E3SM-MMF

5.5.1 KPP Verification Plan

The overarching challenge problem is to develop a multi-scale Earth system model with a weather-resolving global atmosphere (~ 25 km), an embedded cloud-resolving model to explicitly represent convective processes (~ 1 km), an eddy-resolving ocean, and ice components, all while obtaining the necessary throughput to run 10–100 member ensembles of 100 y simulations in less than 1 calendar year.

The challenge problem size has several aspects to achieve:

- Explicit representation of convection in the atmosphere with the MMF approach (~ 1 km grid spacing in both horizontal and vertical directions)
- Weather-resolving resolution in the global atmosphere model ($\sim$ 50–25 km average grid spacing in the horizontal directions with ~ 1 km grid spacing in the vertical, similar to current operational forecasts)
- Eddy-resolving ocean/ice model (minimum 18 km resolution in equatorial regions, decreasing to 6 km in polar regions to capture the reduction in eddy size with decreasing Rossby radius of deformation with $O(100)$ levels in the vertical)

The final aspect is to have sufficient model throughput needed to perform the simulation campaign of the challenge problem in the course of 1 calendar year on the exascale Frontier system. The team's minimum requirement (10 member ensemble of 100 y simulations) requires the ability to run 1000 simulated years in 1 calendar year, which can be achieved with a throughput rate of 2.8 SYPD. Ideally, each ensemble member will run at 2.8 SYPD, but this throughput can also be achieved if $NX = 2.8$, where N is the number of ensemble members that can run on the machine simultaneously, and X is equal to the SYPD performance of each ensemble member.

Long term, the E3SM-MMF will be evaluated by using the E3SM water cycle diagnostics package, developed by the E3SM project and will measure the ability of the model to simulate dozens of key aspects of the Earth's climate. Because of the large natural variability and chaotic nature of atmospheric and ocean dynamics, a rigorous assessment of these metrics requires large ensembles of century-length runs identified in the challenge problem, which require a large INCITE-class computing allocation. For our KPP verification, we will run a single short simulation, configured similar to a climate science simulation campaign. The KPP verification will use much fewer resources and is designed to show that E3SM-MMF can achieve the computational performance and realism needed to complete the challenge problem. The performance will be established with a suite of short 5 day strong-scaling benchmark calculations. The realism of the model will be established with select key diagnostics which can be compared to observations.

Verification Simulations

For our verification runs we will be using an E3SM coupled model configuration. This configuration has active atmosphere, land, ocean and sea ice components. Each component requires many input data sets for initial conditions, boundary conditions and other features such as prescribed emissions, stored on the public E3SM data server and downloaded automatically by the case configuration software. The global model will be using as close to the standard E3SM configurations as possible, with the exception of the many changes brought in by the MMF and minor changes in global model resolution.

All verification will be done at the challenge problem resolution, using the "ne60pg2" or finer ("ne120pg2") atmosphere grid, coupled with the "oRRS18to6" ocean grid, running with an embedded cloud resolving model (1km or finer). This problem will be run for 5 days, without I/O, at as close to the limit of strong scaling as possible (~ 4000 nodes). This is a short run and by itself will not require many resources (14 k node hours), but the scaling studies and work leading up to this final run will require many additional simulations not included in this estimate. For the scientific verification we will perform a second 5 day run with I/O of selected fields with similar cost (14 k node hours).

Simulation Artifacts

For the computational performance verification, only the timing files produced by a successful run are necessary. These files contain the time for the 5 day simulation, which is then converted into our KPP metric, SYPD (simulated years per day). Our KPP FOM is to achieve a $50\times$ increase in SYPD as compared to our baseline (a traditional cloud resolving model run on Titan, achieving 0.011 SYPD). Our $50\times$ FOM goal will be met by achieving 0.56 SYPD. To be able to run our challenge problem (1000 years of simulation in one calendar year) we hope to achieve 2.8 SYPD. We could also meet our challenge problem goal if we can obtain 0.56 SYPD on 20% of Frontier, allowing for an ensemble throughput of 2.8 SYPD.

For the model realism verification, the artifacts are model output of predicted precipitation, temperature and wind at 850 mbar, cloud liquid and ice water paths, and surface fluxes. We will then produce map plots of the output fields mentioned above and compare them to observation data during a similar time of year. The goal is to show that the major features of the Earth's water and energy distributions are present. Qualitative comparison of magnitudes and spatial variability for each field will also be discussed to determine the degree of accuracy.

5.5.2 Early Hardware Status

We have not yet run the full E3SM model on early hardware, but instead focus on individual model components or subcomponents that can be tested independently. These components are (1) the CRM (the cloud resolving "super-parameterization" used within each atmosphere cell), (2) key kernels from the MPAS Ocean component model, and (3) RRTMGP, the radiation package used in the atmosphere model.

CRM

The CRM is the mode expensive component in the E3SM-MMF model and as such as received the most amount of work. We started with the System for Atmospheric Modeling (SAM), and first ported this Fortran code to OpenACC and OpenMP. There remain many issues around compiler robustness for these directive-based approaches, prompting us to rewrite SAM in C++, using the YAKL abstraction layer for CPU and GPU execution. This work has all been applied to the SAM single moment microphysics. One of our project's science goals is to upgrade SAM to two moment microphysics, and to do this we brought in two parameterizations written in C++/kokkos from the parent E3SM project (P3 and SCREAM).

Results from three versions of the code on three different platforms are given in the Table 60. Good performance means for the testbeds is defined as being competitive with V100s, and good performance on the Summit V100s means up to $30\times$ speedup with sufficient workload. The best results are obtained with the C++ code with YAKL. We are close to getting the mixed YAKL/Kokkos code working and expect similar results. We continue to invest in the OpenMP work in order to monitor its maturity as compared to the more disruptive C++ approach.

MPAS Kernels

For the MPAS work, we first ran the entire ocean model on the Spock CPUs, ensuring various build and

Table 60: E3SM-MMF CRM results on Summit and Spock.

	CRM 1mom (Fortran/OpenMP)	CRM 1mom (C++/YAKL)	CRM 2mom (C++/YAKL & Kokkos)
Summit	Runs, 2× slower than C++	Good performance	Good performance
Spock	Cray/ROCm: compiler errors	Good performance	Crashes due to Kokkos

Table 61: E3SM-MMF ocean nested-loop kernel timings (lower is better).

	Summit	Spock
Loop-order 1 (CPU opt) on single CPU core:	4.34	1.84
Loop-order 2 (GPU opt) on single CPU core:	8.27	4.25
Loop-order 1 (CPU opt) on single GPU:	1.07	0.17
Loop-order 2 (GPU opt) on single GPU:	0.054	0.089
Data transfer to/from device:	0.75	0.26

library and MPI issues are resolved. The full model is running well, obtaining a 2.3× speedup per core (3.3× overall) as compared to the Summit CPUs and similar scaling at least for small cases that fit on Spock. For GPU acceleration, we are using Fortran with OpenMP directives. We have extracted several kernels that represent a large portion of the computationally significant work in the ocean model. On Spock, we have been focusing on one of the more difficult nested-loop kernels, where we are exploring multiple refactorings and attempting to maintain good CPU performance and GPU acceleration with a single code base. The results comparing Spock and Summit are given in Table 61.

Spock shows significant improvement (for the best performance option) in both data transfer and CPU performance. On the GPU, the best performance is not yet competitive with the best performance on Summit. However, the Cray compiler does do a significantly better job on Spock when presented with less optimal code (running the CPU-optimized code on the GPU).

We are confident the full MPAS ocean component will soon be running on Spock with working accelerated code that should perform well in the short term with OpenMP directives. Longer term we are planning on porting to C++ using YAKL or Kokkos.

RRTMGP

We have recently completed the port of this component into C++/YAKL. This was the last subcomponent needed to get the E3SM-MMF atmosphere running at its full potential on GPUs. The code was developed and tested on Summit and is responsible for the large increase in our FOM. We have not yet collected performance data for this subcomponent on any of the test beds. But we note that the on-node abstraction is relying completely on YAKL and we expect similar performance as described above with the C++/YAKL CRM.

6. DATA ANALYTICS AND OPTIMIZATION APPLICATIONS

End State: Deliver comprehensive data-driven and science-based computational applications able to provide, through effective exploitation of exascale HPC technologies, breakthrough solutions that yield high-confidence insights and answers to challenges in a selected variety of DOE and non-DOE US government agencies.

The DAO L3 area (Table 62) includes applications that do not rely on approximate solutions to equations that state fundamental physical principles or reduced semiempirical models. Instead, DAO applications exploit predictive capabilities that exploit modern data analysis and ML techniques, complex combinatorial models of data interactions, and constrained, nonlinear algebraic representations of complex systems. These applications target the mission space of DOE and other relevant federal agencies—such as the NIH, the NSF, NASA, and NOAA—identified as high priority per the National Strategic Computing Initiative (NSCI). They include a broad range of application areas and techniques, some of which are only recently coming

Table 62: Summary of supported DAO L4 projects.

WBS number	Short name	Project short description	KPP-X
2.2.4.02	ExaSGD	Reliable and efficient planning of the power grid	KPP-2
2.2.4.03	CANDLE	Accelerate and translate cancer research	KPP-1
2.2.4.04	ExaBiome	Improve understanding of the microbiome	KPP-2
2.2.4.05	ExaFEL	Light source-enabled analysis of molecular structure	KPP-2

into maturity in the context of high-end simulation. As such, they represent greater risk but also significant potential for new discovery.

The principal goal of this activity is to develop ECP applications capable of delivering demonstrable solutions to specific DAO challenge problems on the exascale systems. These applications must target science and energy challenge problems within the mission space of the relevant agency. Another objective is to educate, train, and inform DAO staff on the best practice approaches for exascale application development, including an integration objective of demonstrating, for the DAO area, the vital role and return on investment provided by predictive modeling in delivering on DAO strategic goals. An additional objective is to gather requirements from the DAO applications to help guide the ST and Hardware and Integration (HI) R&D activities.

6.1 ExaSGD

6.1.1 KPP Verification Plan

ExaSGD software stack, illustrated in Fig. 46 includes PowerScenarios which generates load scenarios for renewable wind-based generation. Currently this scenario generation is done up front and the results are saved or moved to HPC architectures for optimization calculations. The run-time software stack that is being developed for KPP-2 calculations includes ExaGO which performs the domain-specific calculations around grid modeling and the derivatives needed for optimization. HiOp is the optimization engine that interfaces with ExaGO to optimize power flow based on input requirements. This stack is built using RAJA/Umpire for portability, Magma dense linear solvers, and STRUMPACK. There is also an MPI orchestration module that facilitates multi-CPU operation.

Mileposts approved by the ECP leadership team describe the steps needed to move ExaSGD forward towards successful KPP-2 execution. Three of these mileposts were scheduled for FY21 (Mileposts 3–5) and are referenced later in § 6.1.2. These FY21 mileposts were aimed at increasing the scalability, GPU utilization, and portability of the ExaSGD software stack towards target architectures. For wider context, the significance of all of our mileposts is described in Table 63.

Formulations are mathematical expressions developed and used by the ExaSGD team to clarify details of the calculations to be run, as summarized in Table 64. The most relevant formulations in FY21 are Formulations 3–5. In Formulation 3, there is a stochastic security constrained alternate current optimal power flow analysis. This analysis ensures optimal power dispatch under different contingencies and multiple weather scenarios.

In Formulation 4, the calculation performed under Formulation 3 is extended across multiple time periods. Stochastic security constrained optimal power flow analysis is carried out for each period and different periods are coupled through ramping constraints. This analysis delivers optimal power dispatch that takes into account how fast different generation units can be ramped up and could enable grid operators to use less conservative strategies.

In Formulation 5, the deterministic base case over multiple time periods is supplemented with a system frequency variable so that the problem can be used to plan the fastest recovery of system frequency to normal after a severe disruption of power grid equipment. The evolution of system frequency from one time period

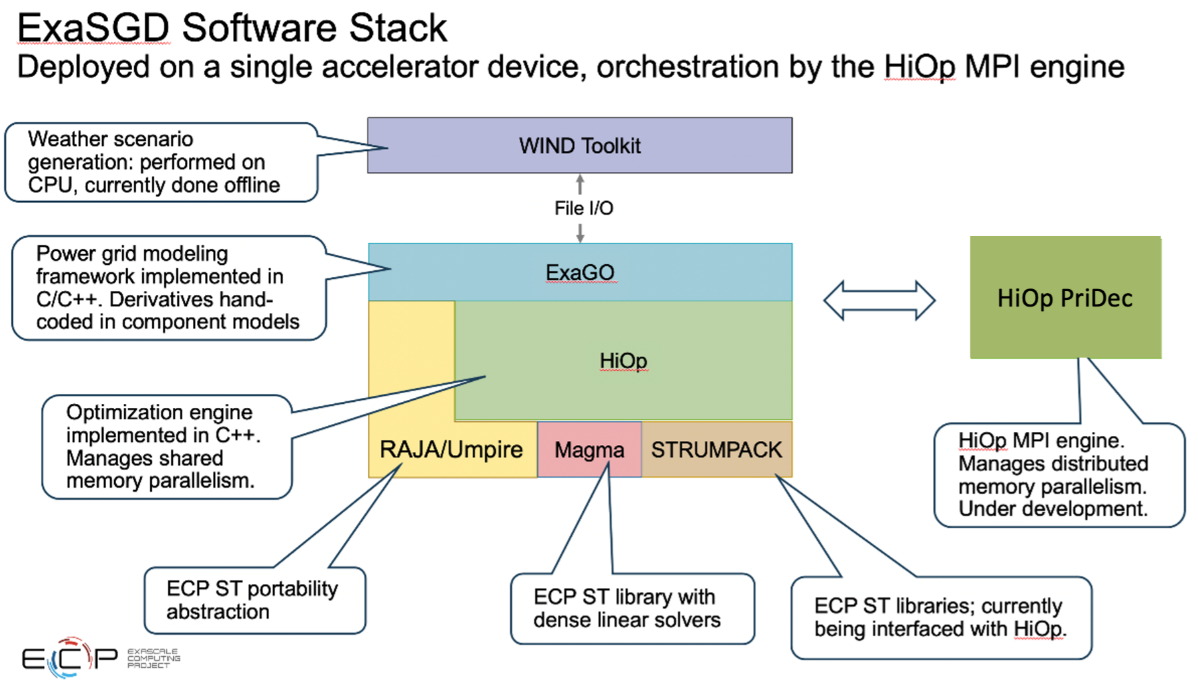

ExaSGD Software Stack
Deployed on a single accelerator device, orchestration by the HiOp MPI engine
Weather scenario generation: performed on CPU, currently done offline
WIND Toolkit
File I/O
Power grid modeling framework implemented in C/C++. Derivatives hand-coded in component models
ExaGO
HiOp PriDec
HiOp
Optimization engine implemented in C++. Manages shared memory parallelism.
RAJA/Umpire
Magma
STRUMPACK
HiOp MPI engine. Manages distributed memory parallelism. Under development.
ECP ST portability abstraction
ECP ST library with dense linear solvers
ECP ST libraries; currently being interfaced with HiOp.

Figure 46: ExaSGD software stack.

Table 63: Mapping of ExaSGD mileposts to KPP-2 progress.

Milepost	Significance
1 (FY20)	Convergence of HiOp (stage 2) on GPU
2 (FY20)	Full software stack integration, additional functionality ported to GPU (1 PFlops)
3 (FY21)	All computing in optimization loops on GPU (10 PFlops)
4 (FY21)	Full stack portability to AMD GPU
5 (FY21)	(stretch) convergence of multiperiod method
6 (FY22)	Scale up on pre Exascale systems (10–100 PFlops)
7 (FY22)	(stretch) scale up multiperiod method
8 (FY22)	Domain relevance-small scale frequency restoration or emergency planning test case
9 (FY23)	KPP-2 Exascale on Frontier
10 (FY23)	Medium-scale frequency restoration or emergency planning

Table 64: Summary of ExaSGD formulations.

Formulation	Temporal Scheme	Multiple Scenarios	Multiple Security Contingencies	Mapping to KPP-2
3	Single period	y	y	Base calculation
4	Multiple time periods	y	y	Multi period stretch goal
5	Includes frequency dynamic	y	y	Frequency restoration stretch goal

Table 65: ExaSGD challenge problem details

Physical phenomena and associated models	A nonlinear optimization model describing physical and control systems for power grid operation
Numerical approach and associated models	Nonlinear optimization with equality and inequality constraints
Simulation details	10^4–10^5 optimization parameters; 10^4–10^5 N-1 and N-k contingencies; 10–100 stochastic scenarios
Implementation details	Individual contingencies and stochastic scenarios are coupled through the base problem but not directly to each other. One power grid model evaluation, such as 2000-bus Texas grid model, is sufficiently sized to occupy high-end GPU resources when using mixed dense-sparse optimization approach. The overall computation size increases by adding different contingencies and stochastic scenarios to the analysis. For 10,000 contingencies and 10 stochastic scenarios, assuming a single grid contingency-scenario instance efficiently utilizes a 10 TFlops GPU device, leads to a $10^4 \times 10 \times 10$ TFlops = EFlops computation.
Demonstration calculation requirements	The software stack implementation depends on the availability of a linear solver that can handle very ill-conditioned problems and achieve high GPU utilization simultaneously. The team has worked with Magma developers to get dense linear solvers that meet all of our functional requirements. Dense linear solvers require model compression in order to be computationally feasible for medium-sized or larger grid models. Several years of wind and solar data are needed to generate stochastic scenarios. Additional data could be required to create cyber scenarios.

to the next is included in the constraints linking the base case operating points in consecutive time periods.

Science Challenge Problem Description

One critical national security challenge is maintaining the integrity of the national power grid under adverse conditions imposed by natural or human-made causes. Having more renewable power sources on the grid, such as solar and wind power, results in a more uncertain power supply, making disruptions even more difficult to manage. Additionally, a large fraction of distributed generation resources (e.g., rooftop solar panels) are behind meter and are not visible to grid operators.

The ExaSGD project is developing algorithms and techniques to address these new challenges and optimize the grid's response to many potential disruption events under different weather scenarios. This may require analyzing thousands of contingencies (i.e. component failures) occurring under thousands of possible and impactful weather scenarios. In this case, a high-fidelity security-constrained optimal power flow analysis requires the simultaneous optimization of millions of power grid realizations within a relatively short turnaround time. This analysis will help grid planners and operators assess alternative grid management strategies to best maintain the integrity of the national power grid under extremely complex and uncertain operating conditions.

The ExaSGD challenge problem is performing stochastic security constrained optimal power flow analysis for day-ahead transmission grid planning for a large grid model. The details of of the challenge problem are given in Table 65.

Verification Simulations

Facility Requirements

In addition to ECP libraries Raja, Umpire, Magma, and PETSc, as well as C++, HIP, and CUDA compilers, the project needs following:

- Storage of $\sim 10\,\text{TB}$ of historical weather data to generate stochastic scenarios.
  - Weather scenarios are currently generated off-line and then passed as processed input (which is MBs in size) to the software stack. This is our backup plan in the event that scenario generation is inefficient or infeasible on the target architectures. At this time, scenario generation takes far more time than the file I/O so this offline preprocessing approach may be what is used in the KPP-2 runs.
- Continuous integration runner on a target hardware platform.
- Ipopt optimization solver with MA57 linear solver from HSL library to create reference/verification solutions for optimal power flow.
  - MA57 licensing issues are handled by Spack because it prompts the users to obtain proprietary HSL code on their own. While proprietary, MA57 is free of charge for individuals for research purposes. A site-wide license for HSL is not required but it would be helpful. HSL is a CPU code and is used on a single node only, mainly for optimization code verification purposes.

Input description and Execution

The challenge problem will be set up as a stochastic security-constrained nonlinear optimal power flow analysis for a 10,000-bus grid model with order of 105 scenario-contingency pairs. The computation will be run on approximately 10,000 GPUs (2500 Frontier nodes). Each contingency-scenario pair will run on a single GPU device and is expected to utilize all of its resources. The contingency/scenario pairs are coupled by an asynchronous MPI engine to the master problem running on a CPU. This approach assumes mixed dense-sparse approach is used. All input data for the KPP-2 computations is already available publicly or is generated by ExaSGD team. To set up the computation, following data is used:

- Power grid models for optimal power flow analysis are available publicly and provided in MATPOWER format (plain text, standard). The target for mixed dense-sparse approach is 10,000-bus US Western Interconnection grid model.
- List of contingencies in custom ExaGO format (plain text, custom). Contingencies are generated for grid models by ExaSGD team assuming single or multiple component failures in the grid. North American Electric Reliability Corporation mandates that transmission grid has to withstand any single component failure (a.k.a. N-1 contingency). In large scale events, such as hurricanes, operators have to consider simultaneous multiple component failures (a.k.a. N-k contingencies).
- Historical weather data in Wind Toolkit format to generate stochastic scenarios.

Simulation Artifacts

Following artifacts will be produced:

- Each contingency/scenario pair will produce a report card, a plain text file containing contingency recourse information. A parser script will process report cards to ensure security constraints are satisfied and to flag those contingencies where partial blackout is the suggested recourse.
- Strong and weak scaling results with respect to number of scenarios/contingencies.
- Performance profiling results showing hardware utilization for GPU devices.

Verification of KPP-2 Threshold

Verification of KPP-2 results will require profiling to verify hardware utilization and verification of solver convergence for given grid model, contingencies and stochastic scenarios. Furthermore, smaller test cases will be validated against commercial tools. Validity of the Exascale result will be extrapolated from the results for smaller test cases. In addition to extrapolation from results for smaller test cases (as large as commercial tools can handle), additional validation will be performed by checking individual scenario-contingency report cards and ensuring that each solution meets security constraints as specified in the analysis input.

Specific verification will include:

1. Strong scaling study for the challenge problem demonstrating scalability of the MPI engine coupling stochastic scenarios and contingencies through the base problem.
2. Profiling of optimal power flow analysis for single contingency on a GPU device showing reasonable hardware utilization (>20 %). Profiling results should be averaged over several different contingencies randomly sampled from the full-scale run.
3. Verification of interior point method convergence for the full-scale problem using standard criteria.

Specific validation activities will include:

1. Validation against commercial tools for problem sizes that commercial tools can handle.
2. Inspection of large-scale results to ensure each scenario/contingency outcome meets security constraints.

Protocol for ensuring that results are reproducible:

- Provide exact version of software used to obtain your results.
  - Tag in the repository or date and branch from which the software stack was built.
- Provide list of dependencies and their versions used to build the software stack.
  - Tag the version of Spack that was used to get the dependencies.
- Provide build command(s) used to build the software stack.
- Provide run commands used to launch the computations and acquire data.

Sparse Linear Solver Stretch Goal

Currently, the mixed dense-sparse approach is the baseline for optimal power flow analysis on GPUs. This approach is used because so far only dense solvers from Magma provide full functionality needed for the analysis and are stable enough to be used in the Exascale context. Sparse linear solvers that meet the analysis requirements and run efficiently on GPUs are not currently available. Even with model compression techniques we use, the mixed dense-sparse approach does not scale well due to $O(N^3)$ complexity of the dense matrix factorization and $O(N^2)$ memory complexity. This limits the size of grid that can be used in the analysis (but not the number of contingencies or stochastic scenarios). Using fully sparse analysis would significantly speed up computations and reduce memory requirements.

In case sparse linear algebra required for this analysis is not available in time, the team will use mixed dense-sparse approach and target grid models with 2000–10,000 buses. With adequate sparse linear algebra available, the team will use largest grid models available (currently 70,000-bus Eastern Interconnection) and will likely require fewer node hours for the large-scale demonstration.

6.1.2 Early Hardware Status

Early Successes

Continuous Integration

ExaSGD builds and tests our key software stack components, HiOp and ExaGO, on three clusters at two facilities on every push, with most or all relevant options enabled, including GPU acceleration. We also have additional low-cost tests that build and test many configurations of our software stack in containers. This automated testing configuration is supported by the Spack package manager, and we are one of the first teams to leverage this Spack capability at scale. We are actively working with the Spack community to develop this capability.

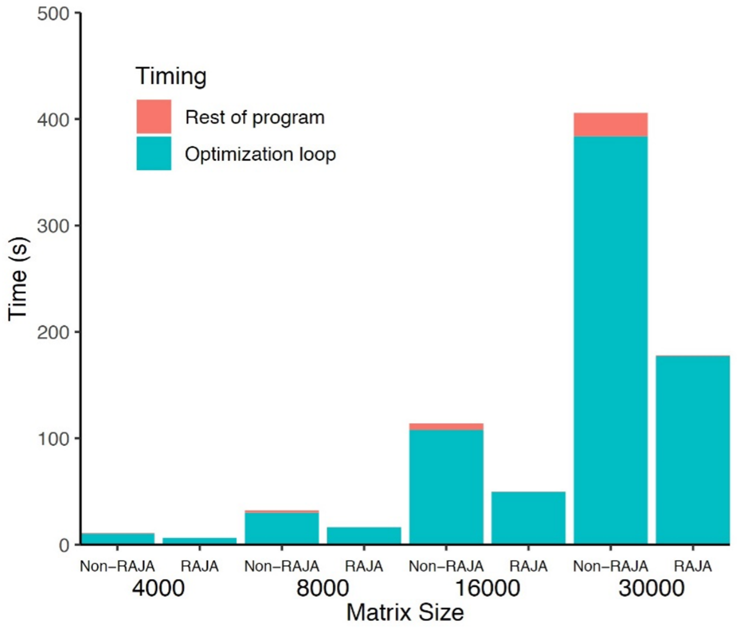


Figure 47: ExaSGD improvement in run time for optimization loop when using RAJA.

We continuously add tests and refine our test suite to match our project milepost objectives. We will be the pilot project for continuous integration on one of the OLCF early access systems where we will build and test our software stack on early access hardware in preparation for Exascale systems. All key software dependencies used in these milepost runs are tested in continuous integration.

Portability with Umpire and RAJA

Calculations for Mileposts 3 and 4 (see Table 63) were run successfully on Summit and Spock, respectively. Nicholson Koukpaizannk and Phil Roth from ORNL were critical to this effort, particularly with the instantiation of our dependencies on Spock with the ROCm HIP compiler toolchain and the Cray provided MPI installation. ORNL collaborators instantiated Umpire and RAJA on Spock and generated modules in our project folder that we were able to use. The changes to our Spack environments are in the ExaSGD Spack repository, and we will continue to incorporate these changes into our Spack packages for HiOp and ExaGO as needed. We instantiated our software stack on Summit twice this year due to the system upgrade. When instantiating after the system upgrade, we experienced an issue with the internal and external version of CUB when building Umpire with the default version of CUDA. Nai-Yuan found a workaround for the milepost run. This issue is documented in the ExaSGD Spack repository and will be investigated further in FY22.

Figure 47 shows early results demonstrating benefits of using RAJA and porting the entire HiOp code to GPUs. In CPU computations roughly 90% of computational cost is in linear solver, therefore the decision to port entire HiOp code to GPU versus only offloading linear solver to GPU was not the clear cut one. In the test results, the problem size is varied from small to the largest problem size that can effectively fit within the GPU memory limit of 16GB on NVIDIA Volta GPU. The timing labeled Non-RAJA corresponds to the best run time obtained by exploiting the GPU for dense linear algebra kernels only. The timing labeled RAJA corresponds to Non-RAJA version further optimized with RAJA-based parallelization of most other operations. The results show significant speedup is obtained if the entire HiOp code is ported to GPU and, more importantly, the speed-up increases with the problem size.

Magma linear solvers

The Magma library provides symmetric indefinite dense linear solvers that meet requirements for nonlinear optimization and that run efficiently on GPUs. Magma provides a solver based on Bunch-Kaufman factorization and a solver using non-pivoting version of LDLT factorization. To date they remain the only stable linear solvers that can be used within interior point optimization on GPUs. The optimization algorithm uses more efficient non-pivoting solver for as long as numerical error is within tolerance. More stable Bunch-Kaufman factorization is used when numerical error needs to be reduced. ExaSGD team has worked with Magma developers to add new functionality to the library, most notably matrix inertia computation, which is used for search direction correction in the optimization algorithm.

Spack

The ExaSGD project has had eight pull requests merged into the official Spack repository. We have continued to update the HiOp and ExaGO packages as we release new versions and our dependencies change. We have also contributed to the Spack info command. We have continued to develop our private Spack packages, and we will continue to merge these changes upstream as they become stable. Users may now install ExaGO and HiOp on many platforms from Spack directly, using only the upstream packages. The upstream HiOp and ExaGO Spack packages now use the built-in CUDA package to determine CUDA dependency information, conforming with Spack best practices. Custom build system logic has been removed in favor of the built-in CUDA base package, and we will do the same with the ROCm base package when our HIP components are more stable.

xSDK compliance

HiOp is almost entirely xSDK compliant. The only remaining required xSDK policy is to have a fully up-to-date Spack package. We have a preliminary Spack package for HiOp. All of HiOp's dependencies are installable through Spack. We have built and installed all of HiOp's dependencies via Spack on a Pacific Northwest National Laboratory (PNNL) cluster (Newell). Aspects of HiOp that were brought into compliance with xSDK include:

1. Version information and library dependencies are now available in HiOp, and the test suite may now be run with valgrind support for leak checks.
2. Cosmin Petra, Slaven Peles and Asher Mancinelli examined HiOp's error checking mechanisms at the interface level and at the level of the linear algebra library. All three reviewers have confirmed the current error handling mechanism is sufficient.
3. We established a preliminary build matrix system to build and test multiple configurations of HiOp on multiple platforms. This testing infrastructure ensures that changes made to conform to xSDK software standards are sustainable. We have committed an initial xSDK compliance document and most of the steps needed to bring HiOp into compliance have already been taken.

Progress against mileposts

In FY21, the ExaSGD team had three mileposts scheduled, 3, 4, and 5 (see Table 63) The following sections describe progress made against successfully completing these mileposts. The goal of these mileposts was to demonstrate integration through the software stack, execution of all inner loop calculations on GPU, and portability across NVIDIA and AMD GPUs on architectures that look increasingly like our final target Exascale architecture. These descriptions are followed by performance profiling results on key parts of the software stack.

Milepost 3—10 PFLOP full-stack run and method convergence on Summit

The description of Milepost 3 is:

> Run Formulation 3 using full ExaSGD software stack with HiOp MPI engine and ExaGO modeling framework on Summit. Use 10,000–20,000 bus grid model with 5+ scenarios, 2000–10,000 contingencies. All computations inside optimization loops run on GPU. Target 10 PFLOP performance.

For this milepost, we executed the ExaSGD software stack on Summit for Formulation 3. The synthetic Texas 2000-bus grid was used rather than our target grids of 10 k–20 k buses because existing compression

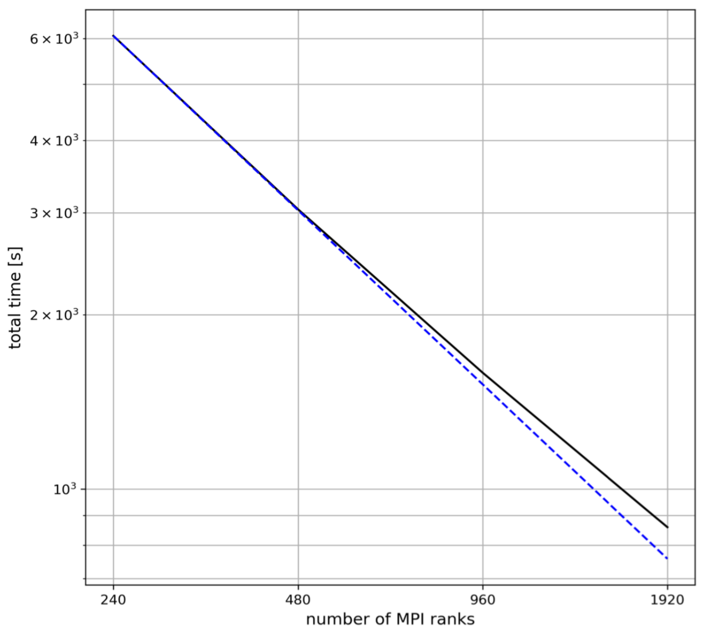


Figure 48: ExaSGD Milepost 3 strong scaling on Summit for the synthetic Texas-2000 bus grid with 1000 contingencies and 10 scenarios.

Table 66: ExaSGD Milepost 4 runtimes for multiple scenarios with varying number of contingencies.

Nodes	Scenarios	Contingencies	Execution time (sec.)
4	5	31	3252
4	5	63	6426
4	5	95	9536

available in ExaGO was insufficient to run 10,000-bus grid model efficiently. This smaller grid was run with 10 wind generation scenarios each having 1000 contingencies. ExaGO's SOPFLOW module was used together with HiOp's primal decomposition solver. All the computations (domain model and optimization kernels) for the second-stage subproblems run on the GPU. The strong scaling of this problem up to 1920 ranks is shown in Fig. 48. It took about 15 minutes for the optimization to converge in two iterations.

Milepost 4—Small scale full-stack run and method convergence on Tulip

The description of Milepost 4 is:

> Run Formulation 3 using full ExaSGD software stack on Tulip. Use 1000–2000 bus grid model with 5+ scenarios and 200–1000 contingencies. All computations inside optimization loops run on AMD GPUs.

For this milepost, we used Spock, instead of Tulip, for our runs. Spock uses AMD GPUs and it has a maximum allocation limit of 4 nodes up to 3 hours. Hence, the number of contingencies was scaled down accordingly to conform to these constraints. We used 5 wind forecast scenarios with increasing number of contingencies, given in Table 66. All the execution runs used 4 nodes on Spock with 4 GPUs each. Similar to Milepost 3, all the second-stage subproblems were fully executed on the GPU.

Milepost 5—Stretch goal: Convergence of a small scale 2-period optimization on Tulip

The description of Milepost 5 is:

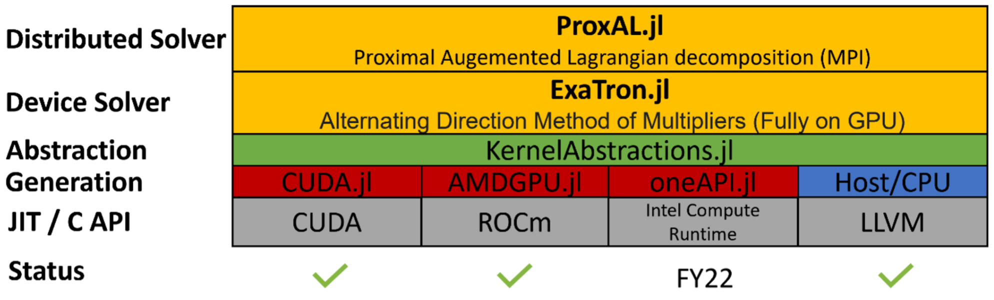
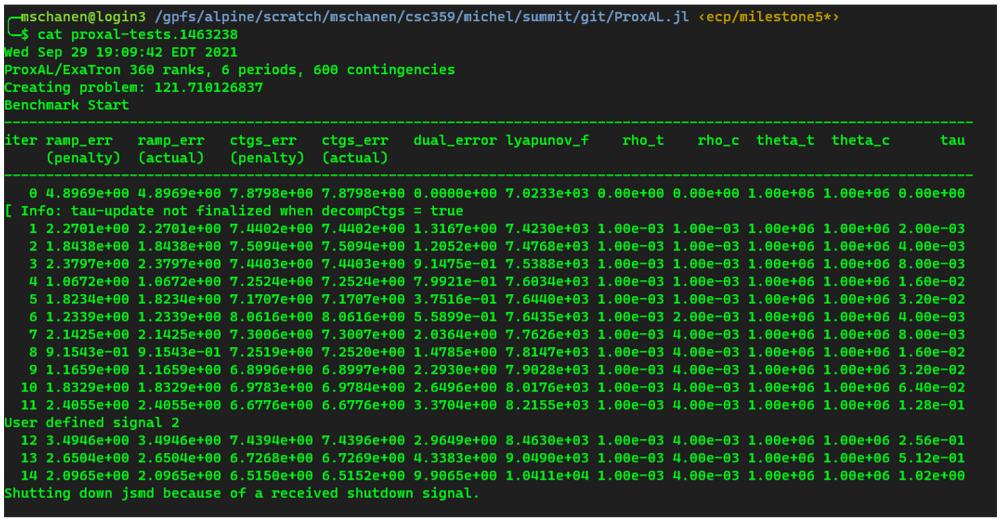


Figure 49: Julia-based software stack for multiple time step ExaSGD stretch goals.



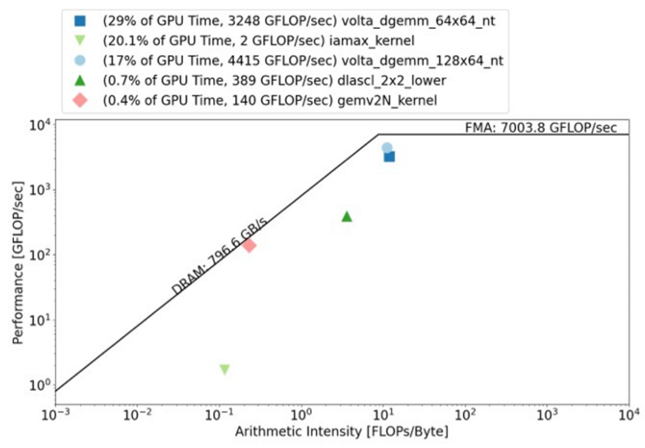
```
┌─mschanen@login3 /gpfs/alpine/scratch/mschanen/csc359/michel/summit/git/ProxAL.jl ‹ecp/milestone5*›
└─$ cat proxal-tests.1463238
Wed Sep 29 19:09:42 EDT 2021
ProxAL/ExaTron 360 ranks, 6 periods, 600 contingencies
Creating problem: 121.710126837
Benchmark Start
-------------------------------------------------------------------------------------------------------------------
iter ramp_err   ramp_err   ctgs_err   ctgs_err   dual_error lyapunov_f    rho_t    rho_c  theta_t  theta_c      tau
     (penalty)  (actual)   (penalty)  (actual)
-------------------------------------------------------------------------------------------------------------------
   0 4.8969e+00 4.8969e+00 7.8798e+00 7.8798e+00 0.0000e+00 7.0233e+03 0.00e+00 0.00e+00 1.00e+06 1.00e+06 0.00e+00
[ Info: tau-update not finalized when decompCtgs = true
   1 2.2701e+00 2.2701e+00 7.4402e+00 7.4402e+00 1.3167e+00 7.4230e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 2.00e-03
   2 1.8438e+00 1.8438e+00 7.5094e+00 7.5094e+00 1.2052e+00 7.4768e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 4.00e-03
   3 2.3797e+00 2.3797e+00 7.4403e+00 7.4403e+00 9.1475e-01 7.5388e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 8.00e-03
   4 1.0672e+00 1.0672e+00 7.2524e+00 7.2524e+00 7.9921e-01 7.6034e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 1.60e-02
   5 1.8234e+00 1.8234e+00 7.1707e+00 7.1707e+00 3.7516e-01 7.6440e+03 1.00e-03 1.00e-03 1.00e+06 1.00e+06 3.20e-02
   6 1.2339e+00 1.2339e+00 8.0616e+00 8.0616e+00 5.5899e-01 7.6435e+03 1.00e-03 2.00e-03 1.00e+06 1.00e+06 4.00e-03
   7 2.1425e+00 2.1425e+00 7.3006e+00 7.3007e+00 2.0364e+00 7.7626e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 8.00e-03
   8 9.1543e-01 9.1543e-01 7.2519e+00 7.2520e+00 1.4785e+00 7.8147e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 1.60e-02
   9 1.1659e+00 1.1659e+00 6.8996e+00 6.8997e+00 2.2930e+00 7.9028e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 3.20e-02
  10 1.8329e+00 1.8329e+00 6.9783e+00 6.9784e+00 2.6496e+00 8.0176e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 6.40e-02
  11 2.4055e+00 2.4055e+00 6.6776e+00 6.6776e+00 3.3704e+00 8.2155e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 1.28e-01
User defined signal 2
  12 3.4946e+00 3.4946e+00 7.4394e+00 7.4396e+00 2.9649e+00 8.4630e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 2.56e-01
  13 2.6504e+00 2.6504e+00 6.7268e+00 6.7269e+00 4.3383e+00 9.0490e+03 1.00e-03 4.00e-03 1.00e+06 1.00e+06 5.12e-01
  14 2.0965e+00 2.0965e+00 6.5150e+00 6.5152e+00 9.9065e+00 1.0411e+04 1.00e-03 4.00e-03 1.00e+06 1.00e+06 1.02e+00
Shutting down jsmd because of a received shutdown signal.
```

Figure 50: `ProxAL.jl`/`ExaTron.jl` on Summit with 600 contingencies/scenarios and 6 periods.

> Run Formulation 4 using full ExaSGD software stack on Tulip and/or Summit. Use 1000 bus grid model with 5+ scenarios and 100–500 contingencies and 2 periods. All computations inside optimization loops run on GPU. The milepost was completed on Spock and Summit using the software stack illustrated in Fig. 49, while only executing a small job on Spock to demonstrate our portability strategy.

For this milepost, Julia was used to rapidly evaluate suitability of different solvers for the multiperiod problem, and to facilitate portability of the resulting code. For the demonstration that we have achieved this milepost, two different codes were run together: 1) ProxAL, which is a primal decomposition driver, served as the orchestrator for the computation; and 2) TRON which runs a Newton-based dense direct linear solver for each execution thread on GPUs. This demonstration used Texas 2 k bus model with $T = 2$ periods and $K = 19$ contingencies. We used 4 Spock nodes (the maximum possible). The version of ExaTron used as a backend here relies on `KernelAbstractions.jl` to run both on CUDA and ROCm architectures. Figure 50 shows the result of this run on Summit with 600 contingencies/scenarios and 6 periods (3600 second-stage problems in total run on 360 GPUs). The configuration was submitted with 1 hour of wall clock time and completed 14 iterations. It may well be possible that this case requires hundreds of iterations. The maximum regular job time for this size is 6 hours.

We ported the high-performance GPU NLP solver `ExaTron.jl` to `KernelAbstractions.jl` allowing us to run both codes using the same code base on two totally different platforms. Both implementations yielded exactly the same result after 99 iterations on the Texas 2 k bus case with 2 periods and 19 contingencies using 4 nodes of Spock. The only difference between the Summit and the Spock run is setting the device



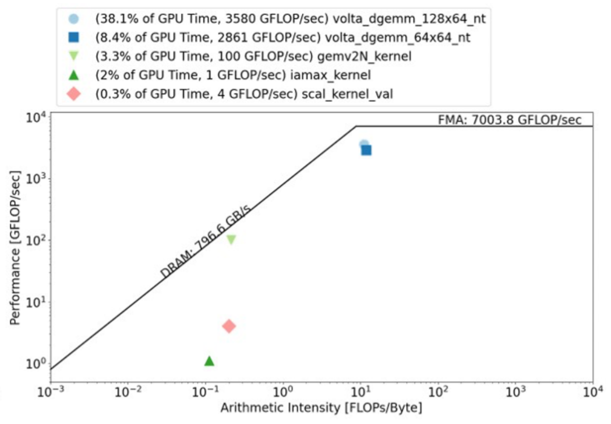


Figure 51: ExaGO GPU roofline analysis of Flops rate for Texas Grid Small (left, 2 k buses) and Medium (right, 10 k buses). Data from Tulip.

from ROCDevice to CUDADevice. The code is exactly the same commit in the repository.

Performance Profiling Results

Profiling was undertaken to understand the performance of single-GPU and multi-GPU hardware utilization as well as the multi-node MPI scalability of the different elements of the ExaSGD software stack for the base challenge problem. Single GPU results illustrated in Fig. 51 demonstrate performance of the 5 most utilized math kernels based on percent of total runtime. Though these only account for $\sim 50\%$ of code run-time during execution, they suggest that beyond arithmetic intensity of 10^{-1} the ExaSGD software stack is achieving reasonably high utilization of GPU's for small and medium-sized problems. For reference, our KPP-2 runs should be about 3–5× the size of medium-scale runs.

Multi-node scalability is largely constrained by blocking in `MPI_Bcast`, but introduces minimal time delays, as illustrated in Fig. 52. This overhead is on the order of 7% of the total execution time for short runs. The team is discussing nonblocking and other asynchronous methods for orchestration to reduce this overhead.

6.2 CANDLE

6.2.1 KPP Verification Plan

CANDLE will execute challenge problems defined by Pilot 1 and Pilot 3. The CANDLE KPP is measured in models trained per unit time for each pilot challenge problem and summarized as the harmonic mean (discussed in detail below).

Verification Simulations

The Pilot 1 challenge problem is akin to a leave-one-out cross validation. The input data is partitioned into n sets and trained for e epochs. The weights are then transferred to initialize the next set of models to be trained, hence the number of models expanding in each iteration. Weights from models at iteration i can be safely (avoiding information leakage) used to initialize model weights for iteration $i+1$, where the training input data in the $i+1$ test set were not in the training set of the model at iteration i.

The Pilot 3 challenge problem focuses on predicting cancer phenotypes and patient treatment trajectories from cancer registry documents. An ensemble of thousands of models will be trained on partitions of a corpus of cancer pathology reports for multi-task classification. To accelerate model training, a hierarchical "partitioned bagging" approach is used which trains ensembles of targeted models on subsets of the corpus, reducing training time and exposing additional parallelism.

The computation differs between the pilot project FOMs in that the neural network architectures differ significantly. The Pilot 1 FOM is based on the Uno model and the Pilot 3 model is based on the MT-HCAN

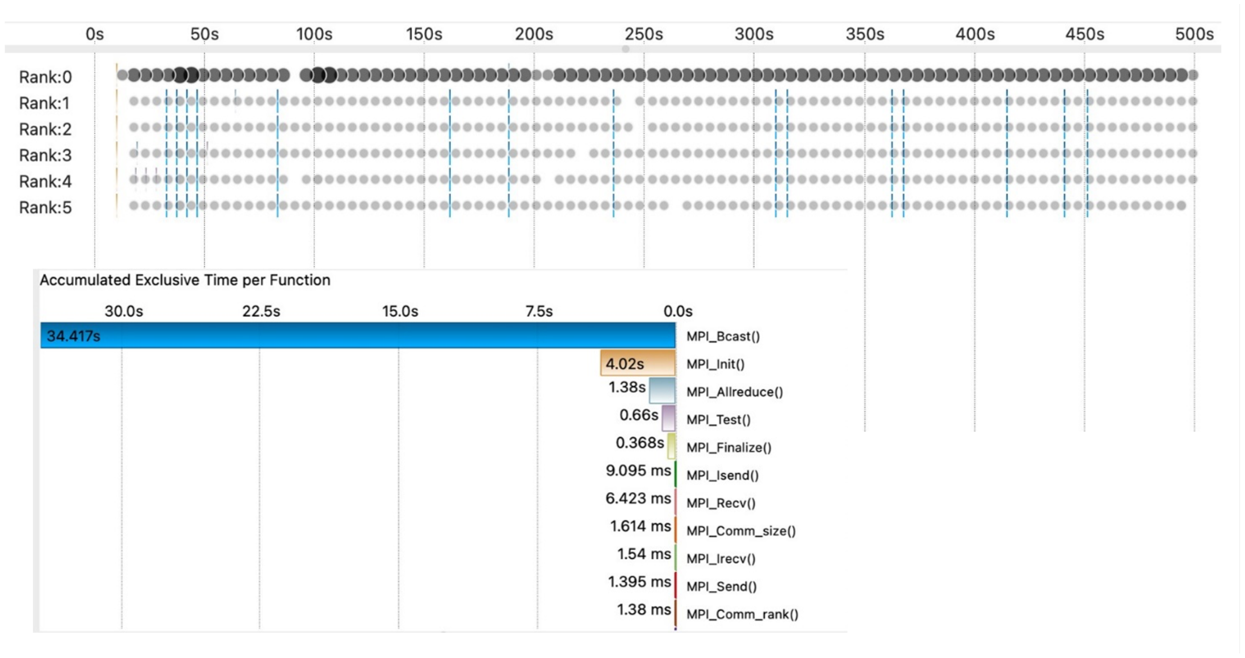


Figure 52: Synchronization across ranks in multi-node execution. `MPI_Bcast()` is the primary consumer of time.

model (also known as P3B4). Combined pilot specific FOMs as a harmonic mean is used to compute the CANDLE FOM.

Uno uses a multi-modal fully connected architecture. Uno implements three feature specific fully-connected layers per feature modality, merges the output of these feature specific layers followed by four fully connected feature agnostic layers. Regularization is achieved using dropout layers throughout the model. Gradients are optimized using the AdaMax algorithm and activation is achieved with the rectified linear activation function.

MT-HCAN uses a multi task hierarchical convolutional architecture with attention. Gradients are updated using the Adam algorithm. Regularization is achieved with dropout layers between the fully connected task specific layers and activation is achieved with the exponential linear unit activation function.

Simulation Artifacts

CANDLE will provide a run summary report of these model training runs that will include CANDLE KPP calculations based on the simulations. We anticipate that the data and models produced by these runs will be analyzed over the subsequent months and that those analyses will result in published reports.

We will do a series of smaller scale runs in preparation for the full scale runs and we will use those runs to determine certain hyperparameters and tune certain partition functions (e.g. number of downstream tasks derived from upstream tasks, etc.). It is anticipated that we will do order 10 jobs at 1–10 % full scale a few jobs at the 25–50 % scale for each of the two tasks. We will include the performance results from these small scale runs as well as the full scale runs.

Verification of KPP-1 Threshold

The primary criteria for determining if the challenge problems run correctly will be to compare model training metrics with previous results. In the case of Uno, the validation R-squared value per epoch will be compared to previous observations. For MT-HCAN, the validation loss per epoch will be compared to previous observations. The comparison of model metrics will be documented along with the KPP calculations.

The KPP performance results will be measured in terms of model instance training throughput and overall effective job throughput as previously described. These will be timing based and our codes are instrumented to such that they emit throughput rates as they run. These are captured in our run logs.

Table 67: Performance of the Uno benchmark on early hardware platforms.

Pilot 1 Uno Benchmark	2021 May (Tulip)		2021 Oct (Spock)	2021 Nov (ATS/JLSE)
	TensorFlow 2.4		TensorFlow 2.5	TensorFlow 2.5
	ROCm 4.0	CUDA	ROCm 4.2.0	Intel oneAPI SDK 2021.3
	MI100	V100	MI100	ATS 80
Epoch1	[redacted]	78	We have measured the performance of the Uno benchmark on Tulip, Spock, and ATS using TensorFlow 2. Using ROCm 4.0 and 4.2.0 on the MI100's and Intel oneAPI SDK 2021.3 on the ATS B0. Results are RSNDA material—available on request.	
Epoch2		79		
Epoch3		77		
Epoch4		78		
Epoch5		77		
Epoch6		79		
Epoch7		76		
Epoch8		77		
Epoch9		77		
Epoch10		77		
Average (s)		77.5		
Throughput (s/s)		5470.3		
Ratio (x/V100)	[redacted]	1.00		

6.2.2 *Early Hardware Status*

The Pilot 1 challenge problem is a unique deep learning workflow that was developed on Summit. In this workflow, the deep learning model is "Uno", a neural network developed in TensorFlow. This network is trained to predict the response of a given tissue cell line to a given drug based on experimental data sets. The workflow divides the total data set into subsets and trains Uno on each subset, reusing trained models as subset sizes increase. This workflow was reported on at MLHPC 2020 [35].

CANDLE has configured the CANDLE/Supervisor workflow environment on Spock. This was straightforward as CANDLE has been run on similar systems. The underlying Swift/T was configured to run with SLURM and the cray-mpich module available on Spock. Python and TensorFlow installations were built with Python 3.7.11, TensorFlow 2.5, and AMD ROCm 4.2.0. The CANDLE libraries and applications were immediately able to run in this environment, demonstrating functional coverage.

CANDLE has accumulated runtime performance numbers on the Uno model on various systems, see Table 67 for more detail. Our current activities include ongoing evaluation of mixed precision on training performance of the deep neural network, and we are looking into the steps to validate the execution of the Pilot 1 challenge problem using model performance metrics (validation R2, validation loss, and validation MSE). At the present time, there appears complete coverage in the underlying libraries for the functionality needed for the Pilot 1 Uno model, and that what remains is performance tuning of the underlying libraries.

In addition to demonstrating the Pilot1 codebase and workflows on the AMD testbeds we have also built the codes and the CANDLE runtime environment on the Intel testbeds and are working on additional performance tuning with Intel engineers in their oneAPI based OneDNN library. Since the ATS testbed is limited in its ability to project performance on to PVC we are using it as primarily a functionality test.

For the Pilot 3, CANDLE has MT-HCAN and Transformers, our models for the CANDLE FOM and stretch goal FOM, respectively, running on Spock. Performance of P3B4 on Spock is consistent with previous measurements on the MI100 nodes of Tulip (Fig. 53). CANDLE anticipates additional optimization when we get access to the MI200s on Crusher, but otherwise are basically ready for Frontier from the perspective of our FOM.

For the stretch goal Transformers, CANDLE has been testing BERT and other Transformer architectures such as GPT-neox on Spock. Per device performance of BERT is modestly better on the MI100 than V100 but not to the extent the relative specs of these GPUs would indicate. Scalability of Transformer pre-training across multiple nodes with data parallelism is adequate and consistent with Spock's network, which is much less capable that expected on Crusher. Library support for the Transformer work on AMD is in a good place, but there are a couple things missing (e.g., a library that we're using for block sparse matrix multiply doesn't

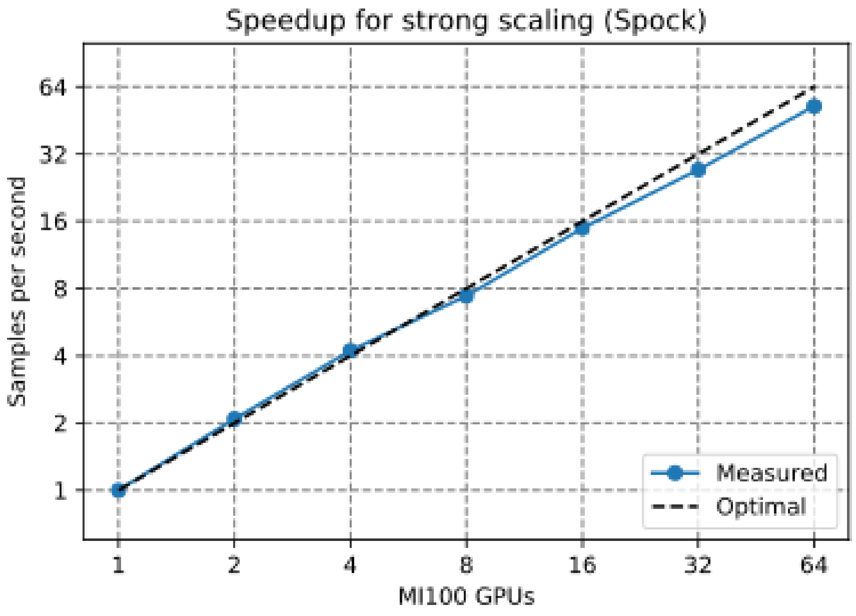


Figure 53: Strong scaling of the P3B4 model on Spock.

yet support HIP) or that don't work as well (mpi4py is having trouble with the MPI configurations from Slurm on Spock). CANDLE doesn't anticipate these being major blockers.

6.3 ExaBiome

6.3.1 KPP Verification Plan

Verification Simulations

Our goal is to assemble a 50 TB dataset. We are still exploring various options for obtaining this dataset that meet the size requirement while also being of scientific interest. Currently there are two primary candidates: part or all of the Human gut microbiome, which is up to 100 TB; and part or all of the Tara oceans dataset, which is 84 TB.

A 50 TB dataset is twice the size of the largest assembly we have performed to date, which is a 25 TB dataset on Summit that took about 45 minutes on 1500 nodes. This was at the limit of the available memory on the Summit GPUs. To successfully assemble the full 50 TB will take at least twice the number of nodes, because of memory requirements alone. However, the complexity of the dataset can have a large impact on the running time, and hence we could well require 6000 Summit-sized nodes for the 50 TB assembly. We anticipate that exascale systems will have more GPU memory per node than Summit, but potential load imbalances in memory usage at larger scales will likely drive up the minimum memory requirements further.

Given these calculations, and the inherent uncertainties in the outcomes, we anticipate that a single run for assembling a 50 TB dataset could use anywhere from 6000 to 9000 nodes on Frontier (which is expected to have at least 9000 nodes), and take half an hour to a couple of hours. In addition, we will need at least 70 TB of storage (50 TB for the inputs and up to 20 TB for the outputs). The exact numbers are dependent on the complexity of the datasets and so we can only provide rough estimates of the requirements.

The 50 TB dataset will likely consist of hundreds of files. These will need to all be transferred to an appropriate scratch file system and, depending on the nature of the parallel file system implementation, striped for maximum I/O performance. For running the assembly, we will use the default settings with MetaHipMer; it should not be necessary to adjust the parameters, except perhaps to cater to any specific aspects of the machine that the application will run on.

In addition to the 50 TB dataset assembly, we will run assemblies of subsets of individual samples from the main dataset. These will be used to calibrate scaling and quality in the final assessment. We will also perform several smaller runs comparing the performance of MetaHipMer with and without GPU acceleration. We have already done so for a 16 TB dataset on Summit and found an overall performance improvement of roughly 1.85× from the use of GPUs. We will repeat a similar experiment on the exascale system to confirm that MetaHipMer is using the GPUs effectively and obtaining performance benefits from them.

Simulation Artifacts

A successful run of MetaHipMer will produce an output directory containing several files. Of these, two are important in verifying that the run completed successfully. The `final_assembly.fasta` is a file containing the final contigs in FASTA format, and the `mhm2.log` file is a record of the run containing the time taken for each stage and the overall time taken (among other data). In addition, the batch scheduler will have logs that record the resources used by a job and the time taken.

Because we will be performing a first-of-its-kind assembly, and many microbial species have never been cultured or sequenced in isolation, we will not have a set of reference genomes to compare against our results. However, there are well-established metrics such as the total contig length, the number of contigs, the percentage of input reads that can align against the assembled contigs, etc.

Verification of KPP-2 Threshold

The KPP-2 threshold will be verified upon successful completion of an assembly of a dataset that contains at least 50 TB of reads. The following points will need to be confirmed:

1. The input dataset totaled at least 50 TB. This will be shown in the MetaHipMer log file (`mhm2.log`) and also can be confirmed by checking the size of the input dataset from the file system.
2. The run completed successfully, which will be indicated by two separate facts: the log showing that the final assembly was completed, and the presence of the final assembly file in the output directory (`final_assembly.fasta`). Furthermore, the number of A,C,G,T,N characters in the final assembly must be the same as the total assembled length of contigs reported in the log file.
3. Assessment of the complexity of the dataset to confirm the runtime is roughly as expected. The complexity can be approximately measured as the number of unique k-mers counted during the assembly, which is reported in the MetaHipMer log file.
4. Assessment that the quality of the resulting assembly is reasonable, as measured by the number of reads that map back to the assembly. This should be a relatively high fraction (greater than 50%) for a high quality assembly. Furthermore, the contiguity of the assembly will be assessed by computing the N50. Both the reads-mapped and N50 are expected to be considerably higher than assembly of any single sample from the dataset.
5. Assessment that the GPUs are effectively utilized to improve performance. This evaluation will be done by comparing the running time for the assembly of a smaller dataset (e.g., 16 TB) both with and without GPU acceleration. To run the full 50 TB assembly without GPUs will use excessive resources.
6. The resources used during the run will be verified from two separate sources. First, the MetaHipMer log file records runtime (including time taken by GPU computations), peak memory usage (both on GPUs and CPUs), and number of nodes and cores used. Second, the batch scheduler will have a record of the job execution that will include runtime, memory usage, and node count.

6.3.2 Early Hardware Status

MetaHipMer on Spock

To date, MetaHipMer has been successfully built with new LLVM compilers available on Spock, but we still have some missing components. The GPU offload of MetaHipMer was written in CUDA because of the nature of algorithms and data types involved; we could not use a higher level or a more portable programming model. CUDA to HIP path was straight forward and allowed us to port the alignment module to AMD GPUs, we can expect the same for kmer analysis module as well. However, the local assembly implementation is more complicated and relies on some CUDA intrinsics that are currently not supported in HIP. We are working with our OLCF liaisons to explore potential solutions. Note that MetaHipMer code is successfully running on Perlmutter, but only up to 64 nodes at this time.

HipMCL on Spock

HipMCL uses fast and custom sparse matrix multiplication libraries as part of its computations. These libraries are written in CUDA and we ported them to HIP to run HipMCL on AMD GPUs in a relatively straightforward manner. Some of the noteworthy issues in the porting process are as follows. We needed to update certain sections of our code written with the assumption that warp size cannot be greater than 32. In addition, the multiplication code makes heavy utilization of masked warp shuffles, which were not supported by the ROCm open source development platform. We provided our own intrinsics for that purpose, which are arguably slower than what the vendor could have provided. Finally, our code has some assembly-level portions which utilize language specific to NVIDIA GPUs. These are rewritten in accordance with the instructions for AMD devices. We have successfully run HipMCL on both AMD MI60 (Tulip) and MI100 GPUs (Spock) and have also run it with multiple MPI tasks with MI100 GPUs. The performance seemed to be competitive with a similar setting relative to NVIDIA GPUs (V100).

MetaHipMer on Arcticus

The most efficient way of porting a CUDA/HIP code to Intel GPUs is via SYCL because of the same low level access and the flexibility the two programming models allow. However, the path to SYCL is not as straightforward as HIP, in particular for the type of codes we have where we rely on some CUDA/HIP intrinsics to obtain performance. We discussed some of these challenges in a paper at P3HPC at SC21. Despite these hurdles, we successfully ported the alignment module to SYCL and were able to run experiments on the Intel GPUs (Intel DevCloud). With our current experience, we are confident that we can also port and run MetaHipMer on Aurora but significant effort will be needed to complete this task. We are also exploring the possibility of leveraging HIPLZ to port MetaHipMer to Aurora. This could potentially require less effort than a SYCL implementation, since it would allow for HIP codes to run on Intel GPUs.

6.4 ExaFEL

6.4.1 KPP Verification Plan

ExaFEL's base challenge problem consists of analyzing X-ray diffraction patterns from the Linac Coherent Light Source (LCLS) to discover details of protein atomic structure with greater accuracy and throughput than achievable with legacy methods. Molecular structure is deduced by synthesizing a 3D electron density map from Fourier coefficients that are deduced from Bragg spot intensities. The novelty of the method is to use information from every pixel and from all images, to derive a set of Fourier coefficients that best fits the data. In the remainder of this write-up, we'll call this approach to nanocrystallography analysis X-ray tracing. This global parameter fit will be performed by gradient descent with the LBFGS (or similar) minimizer. MPI worker ranks will be used for massive parallelization of the gradient calculation.

Verification Simulations

As the demo is not intended to showcase live data collection, the primary raw data will be pre-packaged into container files in a standard format that is optimized for parallel reads. On Frontier, files will be pre-staged to flash memory for efficient intake from flash to GPU, optimally at $1\,\mathrm{TB\,s^{-1}}$.

While the example data are not yet defined, it is possible to present one possible scenario so that the high-performance requirements can be analyzed. A single X-ray crystallography experiment might contain 500,000 diffraction patterns from 100 seconds' data acquisition. The whole demonstration could simulate 20 of these "tranches", representing 2000 seconds of LCLS output. The target would be to process data from each tranche independently in less than 2000 seconds, such that results from the first one are completed before data processing for the last one is started.

The algorithm science will not be finalized until FY23, however we can use a preliminary FY21 benchmark to extrapolate the possible characteristics of the final execution. The benchmark consists of 7500 data images, analyzed on 20-Summit nodes in 2000 seconds. Linear scale up to a 500,000-image tranche would require

1333 nodes. However, in extrapolating to Frontier, the nodes will be more powerful, and also in the meantime we will develop more efficient algorithms. For the sake of estimating the scale, let us assume that only 150 nodes will be needed for execution (a 9-fold improvement over current code on Summit). To process all 20 container files will therefore require 3000 nodes, or about one-third of Frontier. Running all 20 at once would take only 2000 seconds (33 minutes), but we will assume a triangular profile, starting one job every 100 seconds, so the total reserved wall time will require 1 hour.

Simulation Artifacts

The program will output a list of physics parameters that locally generate the size, shape and intensity profile of each Bragg spot measurement, and globally generate the Fourier coefficients for the electron density map. We anticipate the following figures of merit:

- Geometrical fidelity. For strong Bragg spots, how closely does the model predict the actual observed spot position? This is expressed as a root-mean-squared deviation in microns.
- The self-consistency of the Bragg spot intensity profile. For each pixel we define a Z-score: $Z =$ (model intensity-experimental intensity)/$(experimental uncertainty)$. For every Bragg spot, we expect the Z-score over all pixels to be distributed with a mean of zero and std. deviation of 1.
- Precision of the Fourier coefficients. This is expressed as a correlation-coefficient between semi-datasets.
- Accuracy of the Fourier coefficients—how well they describe the reality of the protein structure. This is expressed as a percentage difference between coefficients deduced from data, vs. those predicted from the molecular structure after structure solution.
- Accuracy of the anomalous signal. Here, we will look at the anomalous difference in electron density map peak-height for metal centers such as iron and sulphur, when the data are taken above the metal absorption edge.

Verification of KPP-2 Thresholds

We'll examine the following parameters as evidence that the code utilized exascale resources:

- Fraction of the machine being used (# nodes on Frontier).
- Streaming multiprocessor occupancy. Performance analysis of kernel execution, to assess percent of GPU time utilized for kernel execution. The intention is to pack as much wall time with GPU calculation.
- Roofline analysis for the main GPU kernels, to assess the fraction of theoretical throughput given the limiting memory bandwidth or compute capability. For each kernel we can estimate the number of floating operations performed based on the code, and compare to sustained Flops measured in execution.
- Scaling behavior will be analyzed and bottlenecks identified.

We'll adopt the following minimum criteria to verify the achievement of the challenge problem:

- Verify that the program throughput is able to handle the 5 kHz data collection rate expected for a ~30 min burst at LCLS-II-HE.
- Verify that the X-ray tracing approach achieves more accurate results than traditional methods.

KPPs for Single Particle Imaging Analysis (stretch goal)

The exascale application for single particle imaging reconstruction, spinifel, is to produce detailed protein structure and a movie of the protein functioning at room temperature. This involves determining both the orientation and conformation of individual snapshots. The diffraction data are missing the phase information that also needs to be recovered.

We will demonstrate this exascale capability with a realistic simulated dataset (since experimental datasets of proteins undergoing conformational changes do not yet exist). Similar to X-ray crystallography experiments, the dataset for a single conformation may contain 500,000 diffraction patterns from 100 seconds of data acquisition. Multiple datasets from various time delays will be required for reconstructing a movie of the protein in motion. We will attempt to demonstrate the streaming capabilities from LCLS to the remote facilities.

Multiple instances of Spinifel will be run on each dataset where each instance independently determines the three-dimensional electron density of the protein using the M-TIP algorithm. The best electron density will be chosen from these multiple trials. For large datasets, the computation bottleneck will be in orientation matching which has a time complexity of $O(MN)$ and space complexity of $O(MN)$ where M is the number of model snapshots and N is the number of experiment snapshots. For heterogeneous samples, the time complexity can be even worse $O(KMN)$ where K is the number of conformations. More efficient algorithm development will be required to make computation manageable.

Problem Artifacts

The output of Spinifel consists of 3D electron densities and metrics to measure convergence. Anticipated figures of merit are:

1. Fourier shell correlation (FSC) between two three-dimensional volumes from two halves of the dataset. In general, FSC decreases as a function of resolution from 1 to 0. The resolution where the FSC crosses the 0.143 cutoff determines the resolution of the 3D electron density that is deemed to be close to resolution reported by X-ray crystallography. For simulated datasets, we can calculate the FSC with respect to a reference map with a 0.5 cutoff.

2. Average minimum distance between data and model slices. This is an indicator of the degree of agreement between data and the final model.

3. Root mean square change in the three-dimensional volume compared to the previous iteration. A converged solution must not change over iterations.

Verification of KPP-2 Thresholds

Spinifel KPPs are designed to capture (1) the performance of the experiment, (2) computing performance and utilization, and (3) physics outcome:

1. Time to reconstruct to a certain resolution for rapid feedback during the experiment. In order to provide fast feedback information to the experimenters, the goal here is to complete the reconstruction in the same amount of time it takes to acquire the data.

2. Weak/strong scaling of the application on Frontier to study the time to solution. The goal is to show that Legion is taking advantage of task level parallelism compared to MPI implementation of Spinifel. In addition we could consider roofline analysis of the GPUs to assess our ability to maximize I/O throughput and peak theoretical flops.

3. The reconstructed electron density resolution will be compared to the maximum resolution available at the detector. A good reconstruction algorithm must reach the maximum detector resolution given enough snapshots.

6.4.2 Early Hardware Status

Nanocrystallography

Recent performance results for CCTBX on Summit

As part of the FY21 work, we performed strong scaling studies of the CUDA version of nanoBragg on Summit. These are shown in Fig. 54. We see that we achieve near ideal strong scaling (dashed red line) for most system sizes studied. There is a deviation from the ideal strong scaling for large numbers of nodes (>256), which is recovered when the initialization time is removed (right plot), indicating that deviations from strong scaling are due to initialization and I/O.

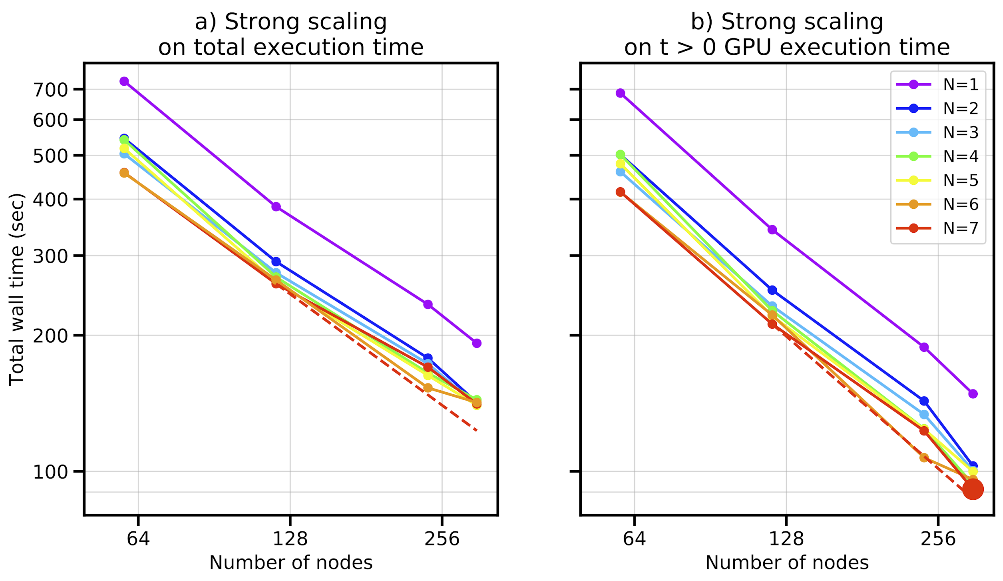


Figure 54: Strong scaling of nanoBragg on Summit. The different colored lines show the wall time for increasing numbers of MPI ranks per GPU (keeping the number of simulated images constant). We find that when multiple MPI ranks share the same GPU, latency hiding decreases the total wall time.

Recent performance results directly comparing CUDA and Kokkos on Perlmutter

We repeated the tests from Summit on Perlmutter. For 64 and 128 nodes, we again tested scenarios where multiple MPI ranks share the same GPU to hide latency, see Fig. 55. We see that the performance of the CUDA version of nanoBragg follows an ideal strong scaling as indicated by the black dashed line. Comparing these results to the strong scaling tests on Summit, we see a performance improvement between 25% to 33%. We note that each node on Perlmutter[5] has 4 × A100 GPUs, while each node on Summit has 6 × V100s. Therefore, our code is able to achieve a significant performance improvement with fewer GPUs.

Comparing the CUDA version with the (refactored) Kokkos version on Perlmutter, we find that the Kokkos version is consistently about 25% faster due to Kokkos' improved memory management, see Fig. 56. These results therefore indicate that our code performs well on new hardware, and that we are not paying a penalty for performance portability (due to Kokkos' improved memory management).

Porting CCTBX on Spock using Kokkos

Using the Kokkos version, we have built and run nanoBragg on Spock. After the initial work on Perlmutter, updating the build process for Spock required only minor changes. We tested the performance of nanoBragg at a small scale by simulating 1200 diffraction images. Using 16 GPUs split over 4 nodes, the simulation took 168 s. This means each GPU needed 2.2 s to simulate one diffraction image. As a comparison, running the same simulation on 4 nodes of Perlmutter took 87 s, corresponding to 1.1 s per image. The original CUDA version of the code required 108 s on Perlmutter with 1.4 s per image.

One problem we encountered is a numerical discrepancy between the reference simulation running purely on the CPU and the AMD MI100 result. While a small difference is to be expected, the relative discrepancy is on the order of 10^{-5}, a hundred times larger than the relative discrepancy for simulations run on NVIDIA's V100 and A100. We are in contact with OLCF staff to investigate the origin of this discrepancy.

We have only ported the forward simulation code to Kokkos, not the first-derivative kernels necessary for diffBragg (stage 1)—that followup work will happen in FY22, within milestone ADSE13-249. Once this is done, we will be in a position to decide if all future GPU work will originate in Kokkos-based code.

Porting CCTBX on Arcticus

[5] All results shown in this section refer to Perlmutter Phase 1.

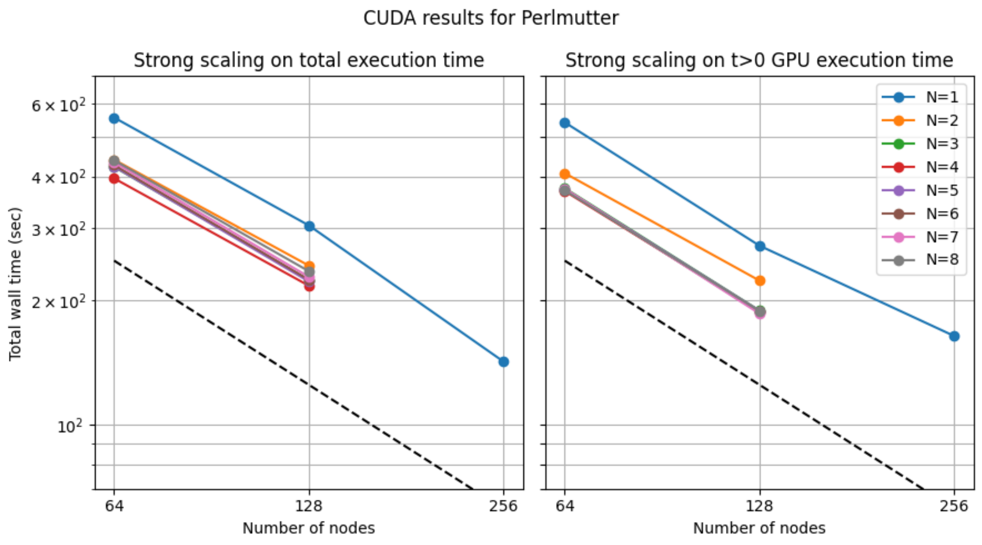


Figure 55: Strong scaling of nanoBragg on Perlmutter using CUDA.

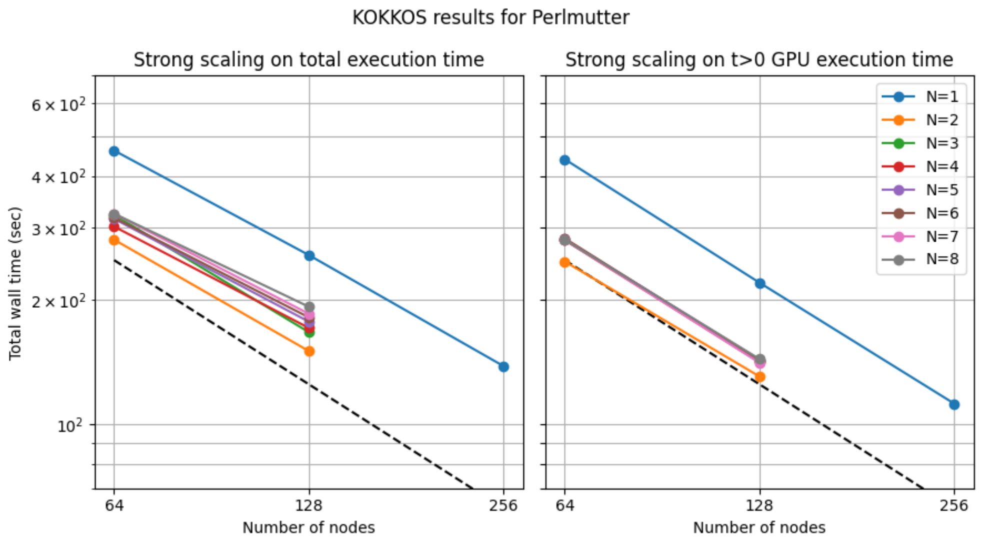
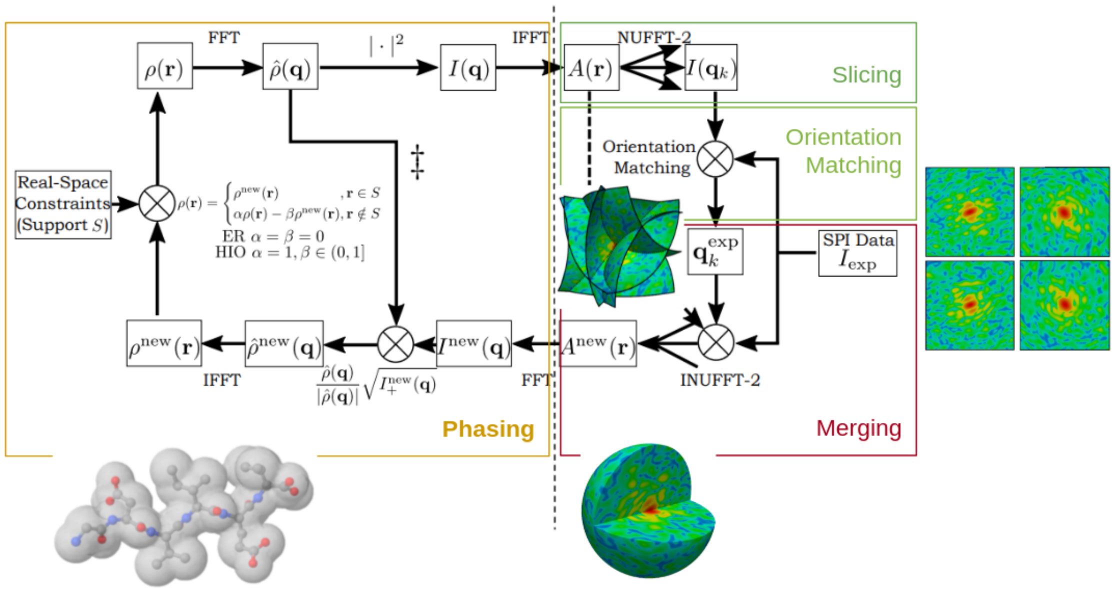


Figure 56: Strong scaling of nanoBragg on Perlmutter using Kokkos.



FFT
$|\cdot|^2$
IFFT
NUFFT-2
$\rho(\mathbf{r})$
$\hat\rho(\mathbf{q})$
$I(\mathbf{q})$
$A(\mathbf{r})$
$I(\mathbf{q}_k)$
Slicing
Orientation Matching
Orientation Matching
Real-Space Constraints (Support S)
$\rho(\mathbf{r}) = \begin{cases}\rho^{new}(\mathbf{r}), & \mathbf{r}\in S\\ \alpha\rho(\mathbf{r})-\beta\rho^{new}(\mathbf{r}), & \mathbf{r}\notin S\end{cases}$
ER $\alpha=\beta=0$
HIO $\alpha=1, \beta\in(0,1]$
$\mathbf{q}_k^{exp}$
SPI Data I_{exp}
$\rho^{new}(\mathbf{r})$
$\hat\rho^{new}(\mathbf{q})$
$I^{new}(\mathbf{q})$
$A^{new}(\mathbf{r})$
IFFT
$\frac{\hat\rho(\mathbf{q})}{|\hat\rho(\mathbf{q})|}\sqrt{I_+^{new}(\mathbf{q})}$
FFT
INUFFT-2
Phasing
Merging

Figure 57: Spinifel is based on the M-TIP (Multi-Tiered Iterative Phasing) algorithm framework that simultaneously determines conformational states, orientations, intensity, and phase from single particle diffraction images.

This work is planned for FY22, milestone ADSE13-249, story ADSE13-242.

Single Particle Imaging (stretch goal)

Spinifel is the ExaFEL application that performs analysis reconstruction for Single Particle Imaging experiments. There are three major components of Spinifel relevant for offloading to GPUs, see Fig. 57:

1. Orientation Matching:
   (a) Slicing: Slicing is a precursor step to matching, where model images are computed via the type-2 (a.k.a. "forward") NUFFT of the autocorrelation of the current electron density estimate.
   (b) Matching: In the matching step, each of the experimental 2D diffraction images are compared to all of the model images. The orientation of each experimental image is taken to be that of the model image that is closest in the Euclidean distance metric.
2. Merging/Autocorrelation Solver: Merging can be formulated as inverting a type-2 NUFFT on the oriented experimental data to solve for the autocorrelation of the electron density estimate, which is equivalent to solving a linear, discrete ill-posed problem of a quadratic form.
3. Phasing: The phasing step converts a 3D diffraction volume to a molecular structure. Starting from a random seed state, repeatedly apply real-space constraints on the electron density and reciprocal-space constraints on its Fourier transform back and forth through the use of FFT and IFFT. In particular, we apply combinations of error reduction (ER), hybrid input—output (HIO) and shrinkwrap algorithms until convergence is reached.

The acceleration approach on Summit was:

- Orientation Matching:
  - Slicing: NUFFT type-2, using cuFINUFFT on NVIDIA GPUs
  - Matching: K nearest neighbor, written in CUDA
- Merging/Autocorrelation Solver: NUFFT type-1, using cuFINUFFT on NVIDIA GPUs; and FFT, using CuPy

- Phasing: FFT, using CuPy

The following sections describe our experience with porting Spinifel to run on the Spock and Arcticus early hardware systems.

Porting Spinifel on Spock

At the time of writing, two of the three major kernels used in these components have been ported to AMD GPUs:

- For orientation matching, the K nearest neighbor kernel was ported to HIP, and has been built and run on AMD GPUs.
- We were able to run phasing on AMD GPUs by using a pre-existing HIP port of CuPy. CuPy is a Python library that supports a NumPy-like interface that runs on GPUs (originally NVIDIA GPUs, but now also AMD GPUs). We built and tested our code with CuPy on Spock. As a part of this process, we discovered one bug in the upstream HIP CuPy implementation, which we reported to the developers (and since has been fixed).

Autocorrelation, and portions of orientation matching, rely on an accelerated NUFFT implementation provided by the cuFINUFFT library. This library is currently written in CUDA. We have done initial work to port this library to HIP. It currently passes a majority of the library's test suite, and work is ongoing to address the remaining test failures and integrate it into Spinifel. In the process of doing this work, we found one bug in HIP that has since been fixed upstream.

Spinifel has two distributed implementations, in MPI and Legion, respectively. Both implementations were straightforward to port to Spock. MPI built out of the box using Spock's provided MPI implementation. Legion was retargeted to Spock using Legion's underlying support for network portability via GASNet. No other changes were required in either system to build and run on Spock's Slingshot network.

The build system for Spinifel has been upgraded to accommodate these components, and builds with AMD GPU support out of the box on Spock.

We tested that Spinifel is able to build and run a sample problem on the system. At this time, we are still waiting to conduct performance experiments, as the majority of the running time goes into the NUFFT operator (via our ported cuFINUFFT), which is not yet fully integrated. We expect to present these results at the review.

Porting Spinifel on Arcticus

As with Spock, we were able to run K nearest neighbor (for orientation matching) with HIP via HIPCL, an implementation of HIP that supports Intel GPUs.

We are planning on running phasing via DPNP, a Python library similar to CuPy that supports Intel GPUs. This work is ongoing, but initial results look promising.

We were assisted in porting the FINUFFT library (CPU version of cuFINUFFT) to Intel GPUs by Argonne Leadership Computing Facility (ALCF) staff via the ExaFEL Technical Execution Plan with ALCF. In the initial work, ANL helped us replace the FFT kernels used internally by FINUFFT with their Intel GPU equivalents. Ongoing work at ALCF is looking at accelerating the remaining portions of the code. We are coordinating with ALCF on the integration of this work with Spinifel, and have successfully built and tested the ported version of the code.

As with Spock, we have run and validated the code, but are waiting to conduct performance experiments until the GPU components are fully integrated. We expect to present these results at the review.

7. CO-DESIGN

End State: Develop cross-cutting, motif-based, co-designed software technologies and integrate them into applications, providing them with the potential to fully use exascale hardware technologies and achieve their challenge problem capabilities. Direct HPC vendors and R&D staff on the key application characteristics that must inform the co-design of exascale software and hardware technologies through proxy application software.

Table 68: Summary of supported CD L4 centers.

WBS number	Short name	Project short description	KPP-X
2.2.6.03	CODAR	CD Center for online data analysis and reduction at the exascale	KPP-3
2.2.6.04	CoPA	CD Center for particle applications	KPP-3
2.2.6.05	AMReX	Block-structured AMR CD Center	KPP-3
2.2.6.06	CEED	Center for efficient exascale discretizations	KPP-3
2.2.6.07	ExaGraph	GraphEx CD center	KPP-3
2.2.6.08	ExaLearn	CD Center for exascale ML technologies	KPP-3

The CD activity includes six CD centers focused on specific computational motifs. These target cross-cutting algorithmic methods that capture the most common patterns of computation and communication, known as *motifs*, in the ECP applications. The current list of motifs includes structured and unstructured grids with AMR, dense and sparse linear algebra, spectral methods, particle methods, MC methods, backtrack/branch-and-bound, combinatorial logic, dynamic programming, finite state machine, graphical models, graph traversal, and map reduce. All of these motifs and others that might emerge over the lifetime of the ECP—including ML methods, the focus of a recently added CD center—will be considered within the CD activity with the exception of dense and sparse linear algebra, which is supported within the ST focus area.

Each of the six funded CD centers (Table 68) focuses on a unique collection of algorithmic motifs needed by two or more applications. The top collections of motifs (based on application requirements) were initially targeted, resulting in corresponding CD centers. The goal of the CD activity is to integrate the rapidly developing software stack with emerging hardware technologies while developing software components that embody the most common application motifs. These co-designed components will then be integrated into the respective application software environments for testing, use, and requirements feedback. This process must balance application requirements with constraints imposed by the hardware and what is feasible in the software stack to facilitate performant exascale applications.

7.1 CODAR

7.1.1 KPP Readiness

The CODAR team has good results to report with three application teams: Fusion Whole Device Modeling (WDM, § 4.5), CANDLE (§ 6.2), and WarpX (§ 4.6). Work with the LatticeQCD (§ 3.1) application team and the ExaLearn (§ 7.6) co-design center is underway.

Whole Device Model Application (WDMApp)

The CODAR team has been working closely with the WDMApp team to implement a coupling framework developed in CODAR. The WDMApp project focuses on developing high-fidelity whole fusion device models by using coupling techniques. It is essential to have tools to manage concurrent runs of two or multiple applications, each of which simulates different spatial regions with varying physics principles. To enable WDMApp's multi-scale multi-physics coupled simulation runs, the CODAR team worked with scientists in the WDMApp project and developed Cheetah/Savanna [36, 37] (workflow engine) and EFFIS [38] (workflow composition and execution) and provided methods for efficient data exchanges.

We reported results from this work multiple publications; physics results are in the Physics of Plasmas journal [39], while the workflow engine and the framework appeared in multiple computer science workshops and journals [36–38, 40].

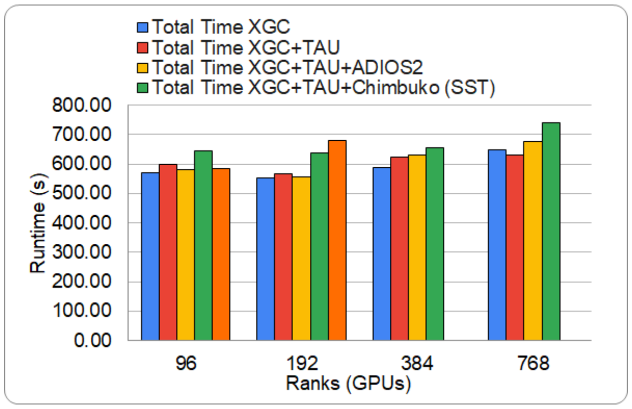


Figure 58: Chimbuko Execution Time Comparison on WDMApp.

The CODAR team also worked with the WDMApp team to evaluate the performance of the XGC plasma physics simulation on Summit. This application is written in C++ and uses both CUDA and Kokkos kernels for GPU computation. Chimbuko was used to analyze runs of XGC up to 768 ranks, focusing primarily on the Chimbuko runtime overhead. Figure 58 demonstrates only minimal overheads when utilizing Chimbuko with XGC providing the BP4 disk-based ADIOS2 transfer method is used to communicate the trace data from TAU to Chimbuko (the SST method uses direct in-memory RDMA communications which can interfere with memory-intensive processes).

At present we are working closely with the XGC team to study in more detail the application traces to isolate and identify any interesting performance anomalies that may impact scaling. We are also beginning to work with the XGC team to analyze performance on Spock.

CANDLE

The CODAR team worked closely with the CANDLE team to apply online analysis techniques to improve performance and scalability of the ML-accelerated molecular dynamics framework DeepDriveMD. DeepDriveMD accelerates large-scale molecular dynamics simulations of biophysical systems (e.g., protein-ligand binding and protein folding) by using ensembles of simulations to explore different parts of conformation space, with ML methods used to select the simulations to perform. A first version of DeepDriveMD repeatedly ran the resulting simulation, ML model training, and ML model inference (for determining new simulations) in sequence. Within each cycle all available processors are used first for the simulation, then a subset is used for training, and finally another subset is used for inference. This SPMD model was simple, but left many processors idle and incurred large I/O costs. The CODAR team worked with CANDLE to develop an entirely new implementation structure in which the simulation, training, and inference steps ran concurrently, on different numbers of processors, with data flowing among them as simulations results were generated, new models were trained, etc., with ADIOS2 used to implement this streaming: see Fig. 59. A paper that reports excellent performance and scientific scaling to 1000 Summit nodes (64,000 cores) has been submitted to the International Parallel and Distributed Processing Symposium. Figure 60 shows some illustrative performance results.

WarpX

We have conducted a comprehensive study of common data layout strategies for parallel I/O operations of ECP applications. We have observed that due to the complex I/O patterns of scientific codes with dynamic load-balancing and/or AMR such as WarpX, no standard data layout strategies used in existing I/O libraries can achieve both satisfactory write and read performance at the same time. Knowing the limitations of existing data layouts, we developed an online data layout reorganization approach, with the aim of enabling good tradeoffs between write and read performance for these complex I/O patterns. This approach can fully reorganize the data layout by leveraging the staging techniques developed by the CODAR team. There are two major benefits brought by this approach. First, due to a high degree of freedom in the reorganization, the read performance can be significantly improved. In particular, the overhead in reading for common read

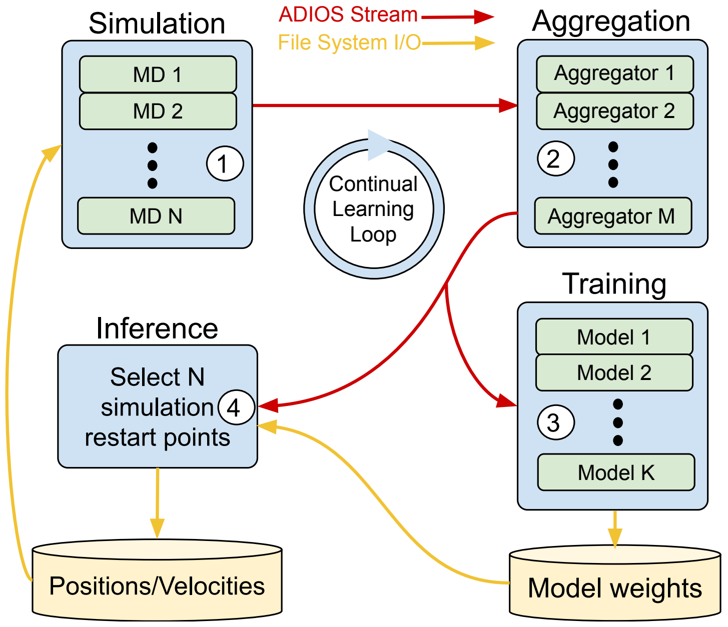


Figure 59: DeepDriveMD architecture. In blue are DeepDriveMD components; green are tasks, managed by the Radical Cyber Tools tools; red are ADIOS streams; yellow is the file system. All tasks run concurrently.

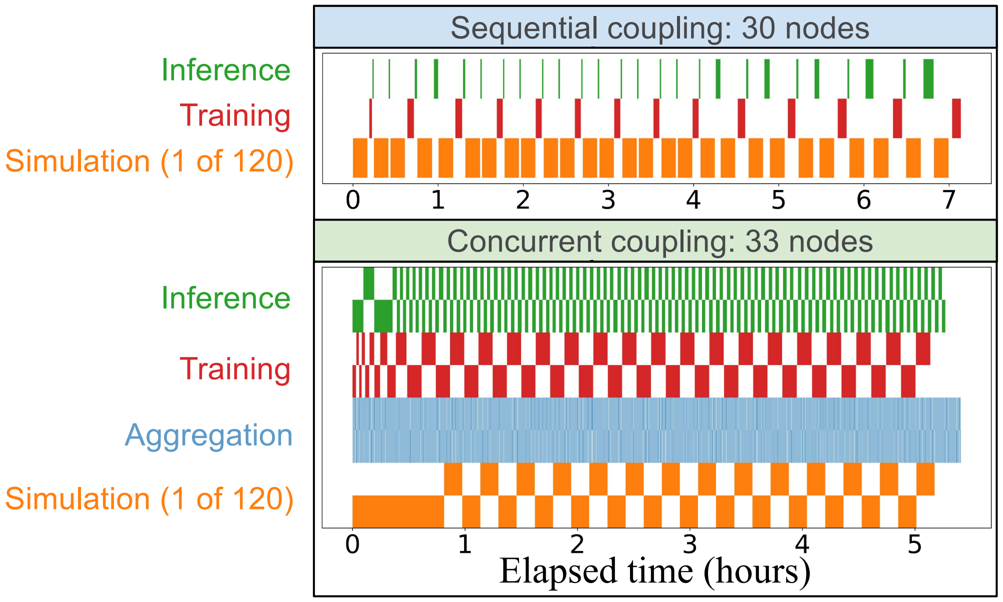


Figure 60: Timeline of task execution when using DeepDriveMD to fold a small protein, BBA, on 30 or 33 nodes of the LLNL Lassen computer. Each bar corresponds to one iteration of a: Simulation (orange); Aggregation (blue); Training (red), and Inference (green) task, respectively. Above: For a version that runs simulation, training, and inference in sequence, with ML run after every second simulation. Below: For a version that the different types of iterations concurrently and without gaps; here, every second iteration is shown on a different line to distinguish between different iterations.

patterns can be reduced by up to 85%. As the outputs of large-scale long-running simulation jobs are usually repeatedly accessed by domain scientists after being generated, this approach will offer a long-term benefit in downstream data workflows. Second, since the data layout reorganization is done in an online fashion through staging, the reorganized data can be accessible to the users almost immediately after the simulation is run. In order to quantify these benefits for real WarpX simulation runs, we also built a model to understand when and how to use the staging-based data layout reorganization can achieve better resource utilization compared to the post-hoc approach.

LatticeQCD

The Grid framework, developed under the ECP program, provides a performance portable library for lattice quantum chromodynamics (QCD) scientific applications that is highly optimized for both many-core CPU SIMD architectures and GPU machines, and is currently deployed at large scale on Perlmutter, Summit and several other large international computer installations. Given that Grid contains Application Programming Interface (API) wrappers for CUDA, oneAPI and HIP as well as taking advantage of numerous modern C++ design principles such as template metaprogramming, it provides an ideal opportunity to develop and demonstrate support for these paradigms within TAU and Chimbuko, and to provide to the Grid team valuable insights into the origin of performance scaling anomalies that are common to modern non-homogeneous supercomputer architectures. In particular, Grid's performance at scale is largely dictated by network and memory bandwidth constraints which are susceptible to local fluctuations caused by resource contention between the application and the operating system, and with other users. We expect Chimbuko's ability to detect and isolate such anomalies to be valuable in understanding Grid's scaling behavior.

Chimbuko relies entirely on the TAU framework to provide application traces for real-time in situ analysis. While TAU has strong support for collecting GPU trace data via the CUDA runtime API, the CPU-side trace collection relies on automatic timer insertion either directly by the C++ compiler or by the PDT tool. Unfortunately the PDT tool does not fully support many modern C++ conventions and is unable to process the Grid source, and compiler instrumentation introduces large runtime overheads due to the inability to intelligently avoid instrumenting short functions. In response to these issues the TAU team have been developing a new C++ instrumentor using the LLVM front-end and testing this explicitly with Grid. Once the tool is made available we will be able to begin studying Grid's performance in more detail.

Exalearn

The CODAR team is working with the Exalearn project to support and analyze the performance of an application for determining optimal experiments by evaluating the impact of model parameters on the uncertainty of the model's description of various physical systems. The specific application evaluates the Kuramoto model of coupled oscillators which is often used to describe chemical and biological systems as well as in neuroscience, using the mean objective cost of uncertainty (MOCU) statistic and compares the result to other statistics. This "MOCU" application is written in Python, using the PyCuda API for computation and the Radical Cybertools (RCT) framework for deployment. The CODAR team has successfully integrated the Chimbuko application into the RCT framework and have demonstrated the collection and analysis of both host and GPU device traces on the Summit machine. At present we are working to analyze the trace of a 120-task run performed on 20 nodes of Summit.

7.1.2 Early Hardware Status

Chimbuko

The CODAR team is working to deploy and test Chimbuko on the Spock system. There are three main components of this work: first, we must ensure the ability to collect GPU and CPU application traces via TAU. To this end we have worked closely with the TAU team to test and aid in the development of TAU's support for the AMD HIP runtime API, particularly with regards to its treatment of the correlation IDs that allow the host and device side function executions to be matched, and have successfully demonstrated collection of complete and well-formed traces from a HIP/C++ mini-app. The second component is the deployment of TAU and Chimbuko alongside all of its dependencies using Spack. This has been our primary difficulty, first in determining how to build a Spack environment that correctly invokes the HIP/clang compilers and couples with the system MPI installation, and in working around apparent bugs in Spack related to incompatibilities between the (incomplete) system Python 3 installation and a complete Python 3 installation built by Spack itself. Finally the deployment of Chimbuko requires careful placement of our in situ analysis component alongside the corresponding rank of the application on the same hardware node, and to dedicate isolated resources to the Chimbuko server processes. To this end we have developed our integration with the Slurm job management system and have demonstrated deployment in the desired fashion. These tasks have now been completed and we have demonstrated a complete online analysis of our GPU mini-app with Chimbuko. As a next stage we plan to work with the XGC and NWChemEx (§ 3.2) teams to analyze the performance of their applications on this system. The CODAR team also aims to test Chimbuko on Arcticus but are awaiting completion of the development of TAU's support for interfacing with the oneAPI runtime for trace data collection.

FTK

We have experimented with compiling and running FTK on Spock. For the compilation, we translated most of the CUDA code into HIP with hipify-perl. We managed to run a synthetic case (2D critical point tracking in time-varying 2048^2 synthetic data) with hardware acceleration; the outputs are correct, and the initial benchmark ran 23× faster with GPU than running on the CPUs for the same number of nodes.

MGARD

We are restructuring the software architecture of MGARD by building parallel abstractions aiming to decouple compression algorithms with hardware-specific implementation details. Supporting a class of CPU/GPU processors is achieved through a corresponding backend that implements basic parallel primitives. Based on our existing CUDA backend, we are close to finishing an implementation of the HIP backend for AMD GPUs. We will develop a SCYL backend for Intel GPUs in the near future. We will have performance results on Spock and Arcticus as the implementations are completed.

Z-Checker

The team recently developed a CUDA version for Z-checker called cuZ-checker [41].

As for the early hardware systems, we have experimented with compiling and running the GPU version of Z-checker (cuZ-checker) on Spock. There are three categories/types of metrics offered by cuZ-checker, which are global reduction based metrics (such as peak signal to noise and maximum compression error), stencil-like calculation metrics (such as derivatives and auto-correlation), and sliding-window based metrics (such as SSIM). We have been able to compute all type-I metrics on Spock's AMD GPU correctly. Calculations of the type-II and type-III metrics work well on CUDA but their HIP version needs further debugging for AMD GPU.

Discussion about GPU versions for oneAPI (Arcticus) is underway.

7.2 CoPA

Progress on CoPA in 2021 is summarized in the context of the libraries—Cabana, PROGRESS/BML, and FFTX and applications partners—EXAALT/LAMMPS/LATTE (§ 3.4), WDMApp/XGC (§ 4.5), ExaAM/PicassoMPM (§ 3.5), and ExaSky/HACC (§ 5.2).

7.2.1 KPP Readiness

EXAALT/LAMMPS/SNAP

CoPA supports the AD KPP-1 EXAALT project via the LAMMPS molecular dynamics code. SNAP performance on a single GPU effectively determines the EXAALT FOM value. The CoPA goal is to make SNAP run as fast as possible on the GPUs from all vendors: AMD, Intel, and NVIDIA. This effort includes re-designs of the SNAP algorithm, Kokkos implementations in LAMMPS, and low-level code optimization both for GPUs generally and vendor hardware specifically. To date, this effort has enabled a current EXAALT FOM of 410× on Summit versus the initial baseline performance of 1× on Mira. New challenges include optimization for AMD and Intel GPUs, as well as design and testing of a neural network SNAP-NN variant.

The integration goals include the following. Use the ExaMiniMD, TestSNAP mini-apps and CabanaMD for initial implementation and benchmarking of re-design and optimization ideas for SNAP as well as other LAMMPS kernels. Implement the successful ideas in LAMMPS itself, mostly within its KOKKOS package, which leverages the Kokkos library (ECP ST project) for portable performance on different GPU hardware. Progress is shown through github merge requests in the public LAMMPS on GitHub (https://github.com/lammps/lammps). The open-source version of LAMMPS is used by EXAALT for its FOM benchmarking.

The target environments are early-Frontier and Frontier architectures. Software environment needs include Kokkos, MPI, cuFFT/rocmFFT, and pytorch. The artifacts to be provided include github merge requests, performance spreadsheet, publications, and the ACM Gordon Bell submission. They confirm the code optimizations and modifications, evolution of the performance, and confirmation though science runs.

EXAALT/LATTE/PROGRESS/BML

CoPA supports the AD KPP-1 EXAALT project via the LATTE Tight-binding quantum molecular dynamics code. CoPA's PROGRESS/BML libraries are being used in EXAALT's stretch goal simulation for a Fission Target Problem, complex defect physics in nuclear fuels. This simulation will show the unravelling of the defect physics controlling performance of nuclear fuels during burnup. High accuracy is required to capture charge transfer around defects and long times to see the defects diffuse. Small systems are sufficient.

The integration goals include the following: (1) Accessing the Exascale architectures without changing the consumer's code through the use of the PROGRESS and BML libraries and (2) increasing performance of the consumer's code (speedup and distributed parallelism). Two different distributed computation strategies have been developed and preliminary implementations are available: (a) A low-level distribution of basic matrix operations implemented in BML and (b) a solver distribution using novel graph-based techniques implemented in PROGRESS. In addition, a compute-intensive "Kernel" method in LATTE is being re-implemented in terms of BML operations to improve performance. The third integration goal is ensuring calculation stability through rigorous testing and benchmarks.

The target environments are early-Frontier and Frontier architectures. Software environment needs include MAGMA, BLAS/Lapack/ScaLAPACK, MPI, and OpenMP (including offload). The artifacts to be provided include github merge requests, input and output from Exascale runs (repository for run scripts, output files, performance plots), publications, and feedback from users. This will serve as confirmation for the code optimizations and modifications, evolution of the performance, and verified though science runs. The PROGRESS and BML libraries are available at https://github.com/lanl/qmd-progress and https://github.com/lanl/bml respectively.

WDMApp/XGC/Cabana

WDMApp will simulate the full-power ITER plasma and predict the height and shape of the edge pedestal, which largely determines the fusion efficiency of the burning plasma. The multiscale edge physics in contact with the odd shaped material wall will modeled by the particle-in-cell code kinetic XGC, while a simplified core kinetic code is coupled to it at a place between core and edge.

XGC will provide milestone reports, including the final report ADSE 12-52, showing integration with CoPA/Cabana; paper and conference presentations showing integration with CoPA/Cabana, exascale runs, including repository for run scripts, output files, and performance plots; and Github releases including merge requests, and ECP Tutorial slides.

The final KPP-1 report ADSE 12-50 will contain the final FOM performance number. The final science report ADSE 12-51 will contain the achievement in the final challenge problem. The XGC Github repository will show how Cabana is utilized in achieving the FOM and challenge problem, with the details presented in ECP tutorial material, conference presentations and papers.

ExaAM/PicassoMPM/Cabana

PicassoMPM is being used with ExaAM application development center to run higher fidelity powder resolved, full physics simulations of melt pool dynamics under laser powder bed fusion additive manufacturing conditions. This includes thermal history and phase evolution for a complex, non-uniform powder bed, providing detailed information to the finite volume thermal history application (OpenFOAM) and to the microstructure evolution application (ExaCA).

Table 69: Status and performance on Spock for the libraries and applications.

Library/App	Status	Performance
PROGRESS/BML	Modifications required to build/run. Complex arithmetic issues—fixed. For offload, `firstprivate` required workarounds.	Using hipmagma builds, full diagonalization is much slower than on Summit. Using hipmagma builds, the SP2 algorithm is comparable to running on Summit.
Cabana	Build and runs with no significant issues.	Performance benchmark suite shows mixed results. Issues with atomics and cache size.
FFTX	Builds and runs.	Using Spiral code generation outperforms hipfft on 3D complex-to-complex FFT.
LAMMPS/SNAP	Builds and runs.	Issues with L1 cache size for MI100.
XGC	Building and running and all correctness tests are passing.	Poor performance of atomics.

The integration goals include CoPA/Cabana (https://github.com/ECP-copa/Cabana) providing all underlying particle and particle-grid capabilities for PicassoMPM. Built on Kokkos, this provides the performance portability with particle thread-parallel iteration, grid thread-parallel iteration, and MPI halo communication.

Artifacts include CoPA milestone reports and publications (including GPU utilization and performance plots on exascale architectures) documenting the Cabana code required to support the ExaAM simulations as well as milestone reports, publications, run scripts and output files showing the ExaAM exascale simulations. The code is in development and available on github in pre-release (https://github.com/picassodev/picasso) Progress is shown through github merge requests. A Cabana tutorial was given during the 2021 ECP annual meeting highlighting the Picasso MPM code.

AMReX/Cabana algorithms

This CoPA integration involves the incorporation of algorithms and strategies from the Cabana library into the AMReX particle library to improve performance of AMReX-based applications on pre-exascale and exascale architectures. AMReX supports a wide variety of ECP applications including WarpX and MFIX which use hybrid particle-grid algorithms along with AMR capabilities. Algorithms for these types of applications, such as particle redistribution and particle halo exchange, were developed by the Cabana team and have been implemented and are available in the AMReX framework.

The goals of the collaborative effort include the following: (1) Improve the performance of AMReX's particle communication routines on pre-exascale and exascale architectures by implementing Cabana algorithms for particle redistribution and halo exchange in AMReX, (2) Demonstrate improvements in MPI scaling performance compared to original AMReX implementation, and (3) Deploy new communication algorithms in MFIX and other AMReX-based ECP applications.

Artifacts include a github link to code implementation of algorithms in AMReX, a letter from AMReX client, slides from previous review showing performance plot, and updated performance slides from exascale runs using these capabilities in AMReX.

7.2.2 Early Hardware Status

In this section we show the current status and performance, where available, for the CoPA libraries and applications on early exascale architectures. Table 69 summarizes the status and performance on Spock. Libraries and applications were able to build and run, though some issues were encountered. Performance is still lacking with the current version of the hardware. Table 70 summarizes the status and performance on Crusher. Libraries and applications were able to build and run, but performance in some cases is lacking.

PROGRESS/BML

The BML library was build on Spock using the CRAY SDK. Complex arithmetic issues were found and

Table 70: Status and performance on Crusher for the libraries and applications.

Library/App	Status	Performance
PROGRESS/BML	Builds and runs.	Performance for diagonalization and SP2 are 2× faster than on Spock.
Cabana	Builds and runs.	Allocations and communication are faster, but iteration is slower.
FFTX	Builds and runs.	On the sizes tested, FFTX and rocFFT were comparable.
LAMMPS/SNAP	Builds and runs.	Algorithmic improvements have led to 24× speedup since our baseline version. Smaller L1 cache is still an issue.
XGC	Builds and runs.	Individual Kernels show a small performance improvement compared to Summit. In a full XGC case, 64 nodes performed 2.3× slower than 256 Summit nodes at similar theoretical peak FLOPS.

reported on the Frontier Confluence site, and have been fixed. The `firstprivate` clause is not supported for offload, requiring the following workaround.

- A pool of shared working arrays was defined
- The work was subdivided into chunks by introducing a new outer `omp parallel for` loop with dimension `OFFLOAD_NUM_CHUNKS` (new macro with default value provided in macros.h)
- Although the working arrays are shared, each thread accesses its own unique set of arrays, avoiding race conditions
- See e.g., https://github.com/lanl/bml/blob/master/src/C-interface/ellpack/bml_multiply_ellpack_typed.c

The same workaround is needed for the Intel SDK, and similar OpenMP issues have existed for years for IBM.

Using the consultant hipmagma builds on Spock, diagonalization and the SP2 algorithm performance are shown for Summit and Spock in Fig. 61. For diagonalization, results on Spock are much slower than Summit, while the SP2 performance is comparable on both.

Performance of diagonalization and the SP2 algorithm are shown for Spock and Crusher in Fig. 62. On Crusher with AMD MI250X GPU, dense soft matter benchmarks show that the diagonalization solver is 2× faster than Spock (and 3× slower than on NVIDIA A100 using cuSolver for large matrices). The SP2 solver on Crusher is 2× faster than Spock (and comparable to NVIDIA A100). MAGMA was used for all SP2 runs.

On Crusher, rocSolver is available but is prohibitively slow compared to MAGMA. AMD is working on this.

Cabana

The Cabana performance benchmark suite consists of the following routines, which include initialization and iteration for each case:

- Sorting: creating the permutation vector and permuting the array
- Neighbor list: building the neighbor list and traversing the list
- Communication: determining the destinations nodes and sending

Results are shown in Fig. 63 for the neighbor list performance benchmark. Overall, the MI100 does better with more work (usually outperforming V100 at some point) for enough particles (or neighbors). Communication is markedly improved on Spock, but we need more tests to determine if that's the network

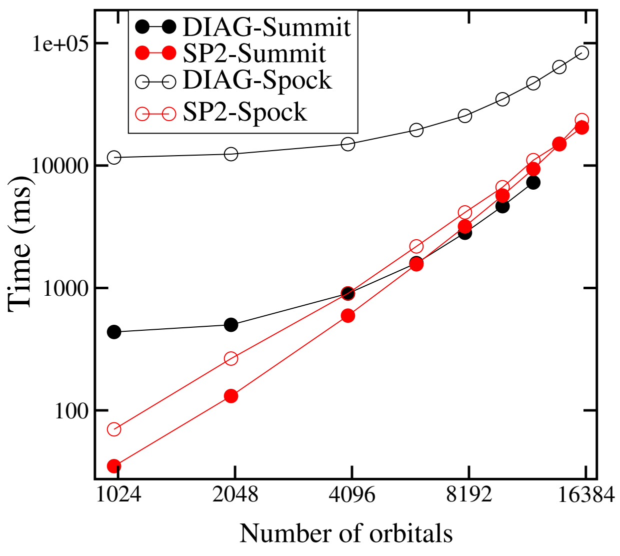


Figure 61: BML performance comparison for diagonalization and the SP2 algorithm on Spock versus Summit.

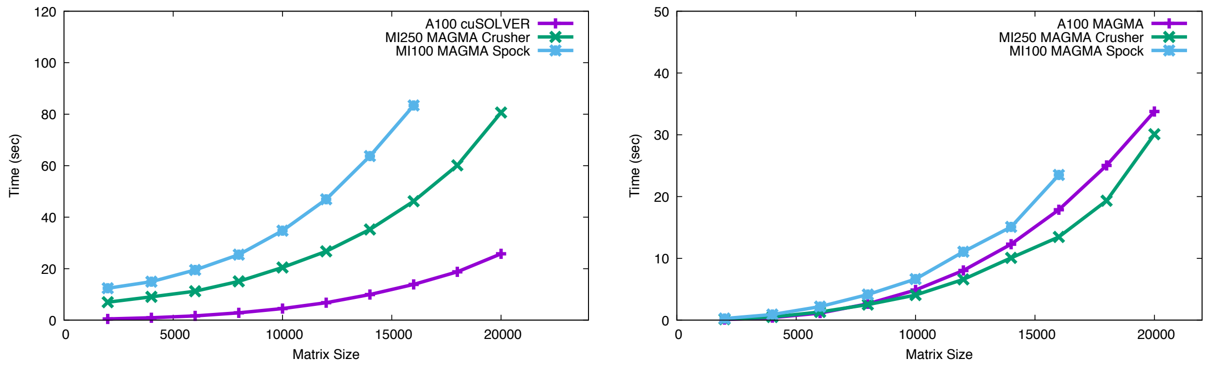


Figure 62: BML performance comparison for diagonalization (first panel) and the SP2 algorithm (second panel) on Crusher (with AMD MI250X, green) and Spock (with AMD MI100, cyan) and NVIDIA A100 GPU (purple).

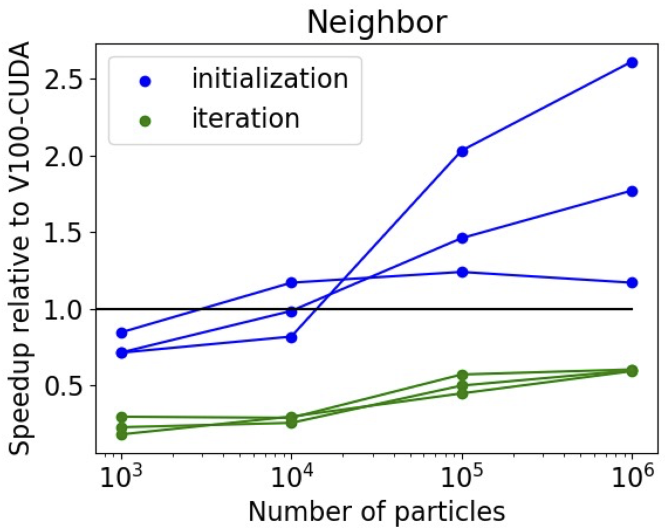


Figure 63: Performance comparison for Cabana initialization and iteration benchmarks on Spock versus Summit for the neighbor list benchmark.

itself, the GPU communication, relative CPU performance, etc. For runs that are not as performant on Spock the lack of hardware atomics and small cache sizes are the most likely issues, requiring more investigation.

As an example in which all of the benchmark routines are used in a Cabana-based code, results are shown in Fig. 64 comparing 1 NVIDIA V100 GPU versus 1 AMD MI100 GPU versus 42 IBM Power9 CPUs versus 64 AMD EPYC CPUs.

As seen in Fig. 65, Cabana is faster on Crusher AMD MI250X than NVIDIA V100 for some kernels and generally for larger system sizes tested in benchmarks. Allocations and communication are noticeably faster, but iteration (especially small sizes) are slower. Fig. 66 shows performance of the CabanaMD proxy app for 3 system sizes: 256k atoms, 2M atoms, and 8M atoms. Forces are slightly slower for AMD MI250X, but communication is slightly faster, resulting in similar run times.

FFTX

In Fig. 67, on Spock we see that hipfft (red) has highly non-uniform performance as a function of the size of the FFT transform (10× variation). The fastest cases violate documented API (read-only input). FFTX performance matches the best performance of hipfft, with more uniform behavior and is within 2× of the roofline limit.

In Fig. 68, on Crusher we ran rocFFT to compare against FFTX for 18 different sizes. rocFFT in rocm 4.5.0 consistently crashes when creating a plan for a 3D FFT of dimensions 128 x 128 x 680. On the sizes tested, rocFFT and FFTX gave results that agreed with each other. Depending on the size, either rocFFT or FFTX is faster, a half and half split.

For 18 different 3D sizes, we ran real-to-complex, complex-to-real, and complex-to-complex forward and inverse FFTs on random inputs on AMD MI250X GPUs using both FFTX and rocFFT, and compared the answers as well as the timings. The answers all agreed to within roundoff error. The default version, rocm 4.5.0 was used on Crusher.

Fig. 68 shows inferred gigaflops per second for 3D complex-to-complex FFT on Crusher, for FFTX (blue crosses) and rocFFT (red circles), and maximum from a roofline model (black stars) that assumes arithmetic intensity of one flop per byte, 16 bytes per complex number, 3 round-trips to GPU fast memory, and maximum bandwidth of 1.6 TB/sec. These flop rates are all inferred from an estimate of 5 N log2(N) flops for a domain of N points, and averaging run times over all but the first of 20 iterations.

LAMMPS/SNAP

On a V100 GPU unified cache is shared between L1 and shared memory, 128 kB total per SM (80 SM

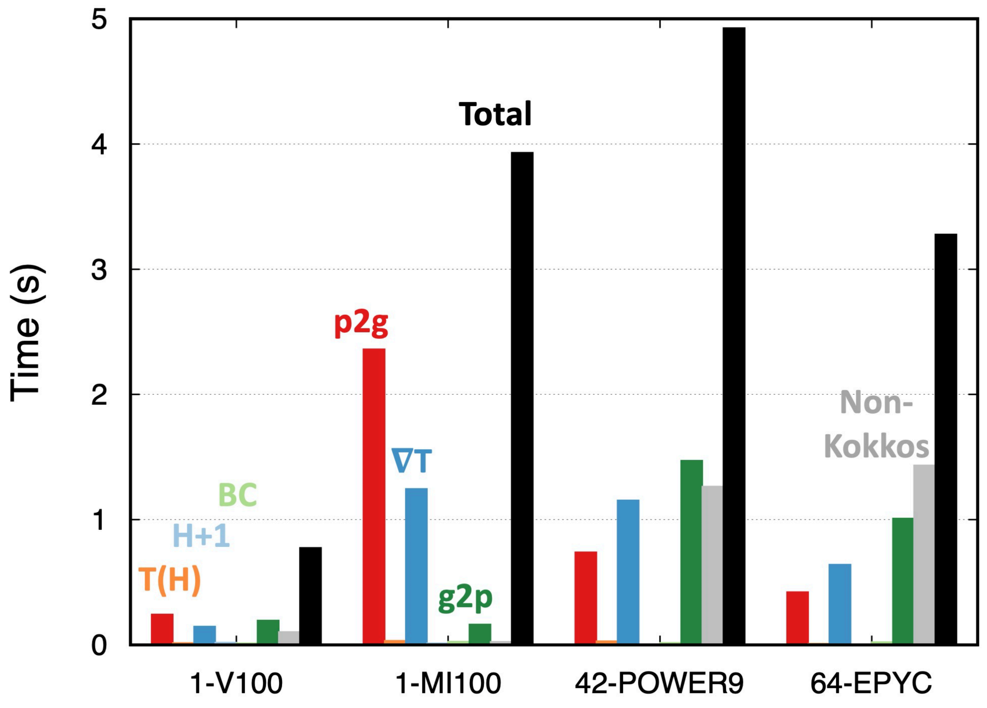


Figure 64: Time-to-solution comparing 1 NVIDIA V100 GPU versus 1 AMD MI100 GPU versus 42 IBM Power9 CPUs versus 64 AMD EPYC CPUs for the Cabana-based PicassoMPM code.

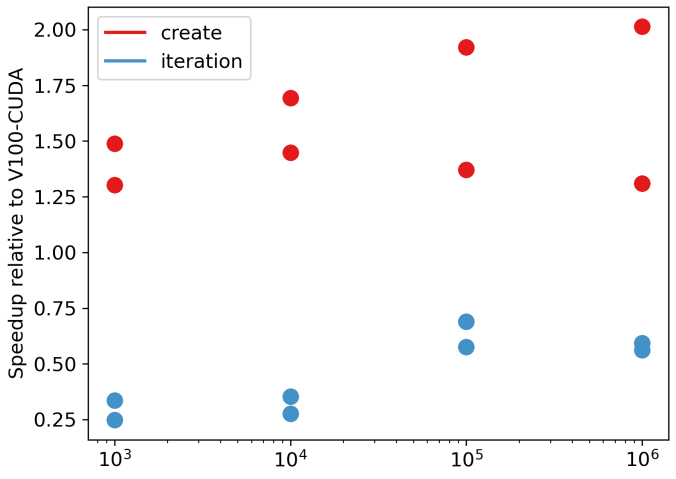


Figure 65: Performance comparison for Cabana initialization and iteration benchmarks on Crusher versus Summit for the neighbor list benchmark.

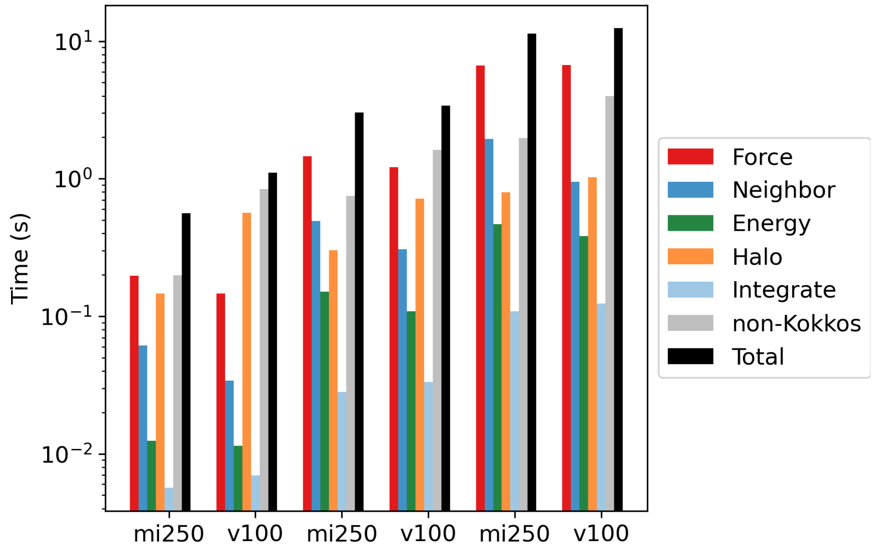


Figure 66: Time-to-solution comparing 1 NVIDIA V100 GPU versus 1 AMD MI250X GPU versus 42 IBM Power9 CPUs versus 64 AMD EPYC CPUs for the CabanaMD proxy app for 3 different system sizes.

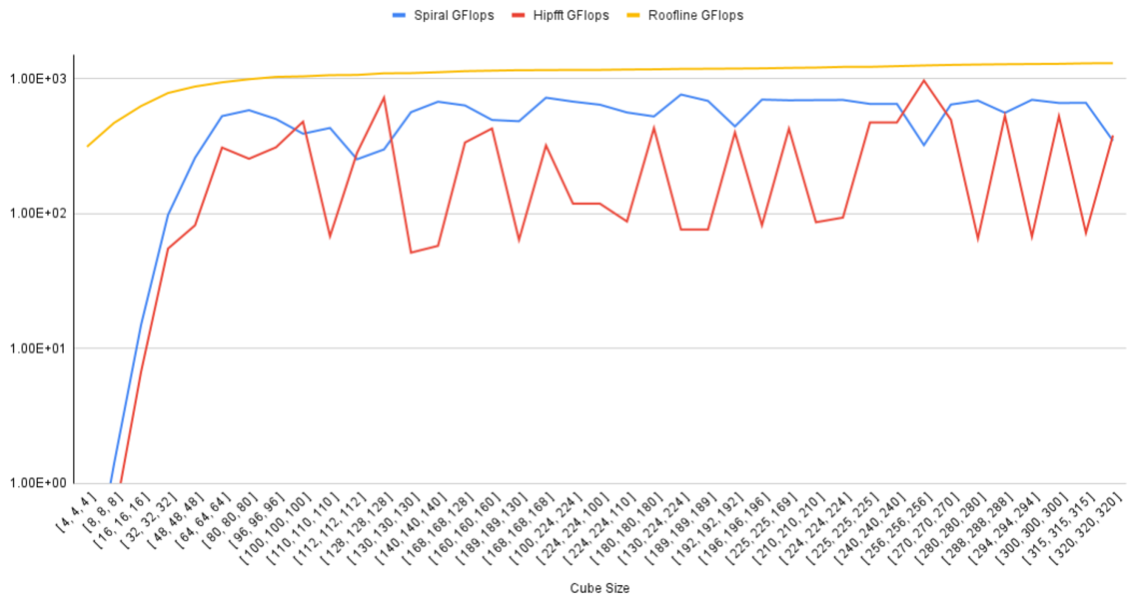


Figure 67: Performance of FFTX with Spiral code generation (blue) compared to hipfft (red) on Spock. FFTX matches the best performance of hipfft, with more uniform behavior and is within 2× of the roofline limit (yellow).

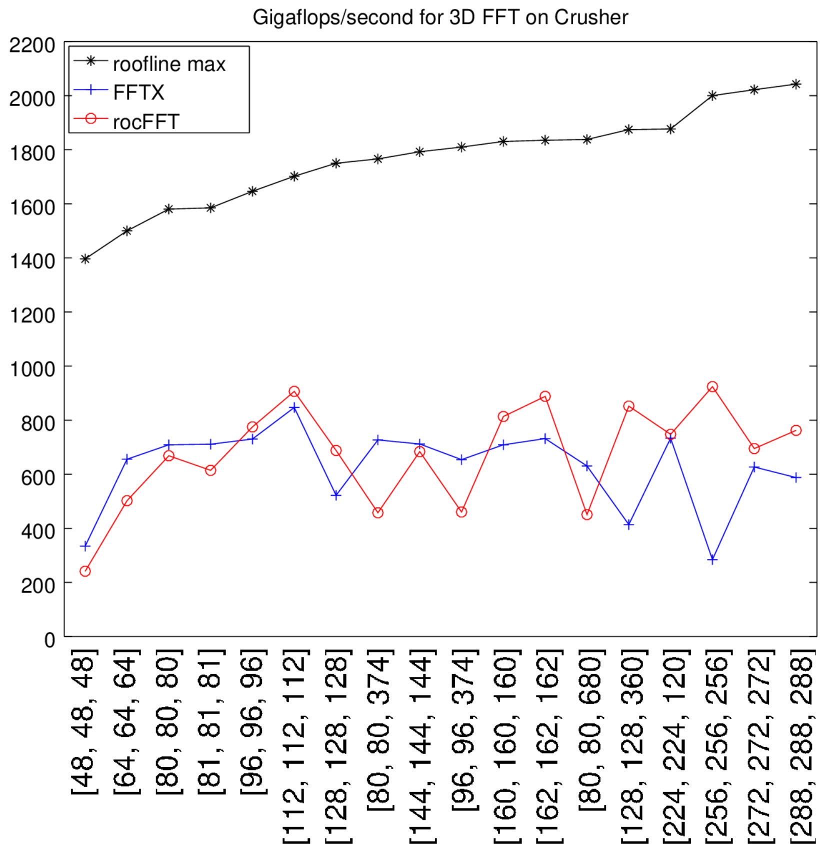


Figure 68: Performance of FFTX with Spiral code generation (blue) compared to rocFFT (red) on Crusher. FFTX and rocFFT gave comparable results, each is faster for half the sizes.

Table 71: Performance of SNAP is shown as katom steps s^{-1} on a single GPU of different types (V100, A100, MI100).

Hardware (Kokkos backend)	TestSNAP	LAMMPS	LAMMPS/V100
Summit V100 (CUDA)	124	132	1.0
Spock MI100 (HIP)	63	55	0.42
Perlmutter A100 (CUDA)	179	174	1.32

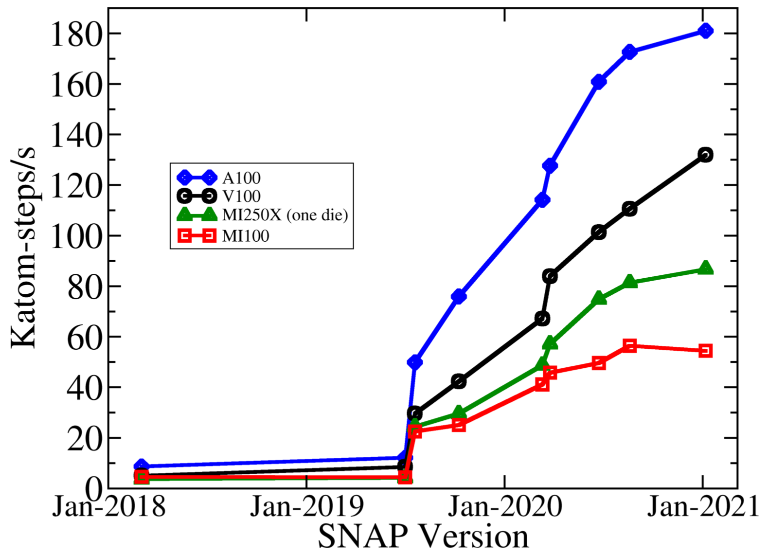


Figure 69: Performance of the SNAP potential over time.

total). SNAP runs with 96 kB L1-cache. On the MI100 GPU there is 16 kB fixed L1 cache and a separate 64 kB shared memory per SM. It appears that the 6× smaller (per SM) L1 cache on MI100 is greatly hurting SNAP performance relative to the V100.

The performance on the SNAP benchmark for EXAALT on a Spock MI100 GPU is 2.4× slower than a Summit V100 GPU as shown in Table 71. These are 3 current obstacles to figuring out why that is, and hopefully fixing it.

1. Unlike for the V100, there is no tool available that we know of to measure the L1-cache hit rate on the MI100. This would help us diagnose the performance gap.

2. If the performance difference is mostly due to a much smaller L1 cache on MI100 versus V100, are there workarounds for this? Will the cache size change on future AMD GPU generations?

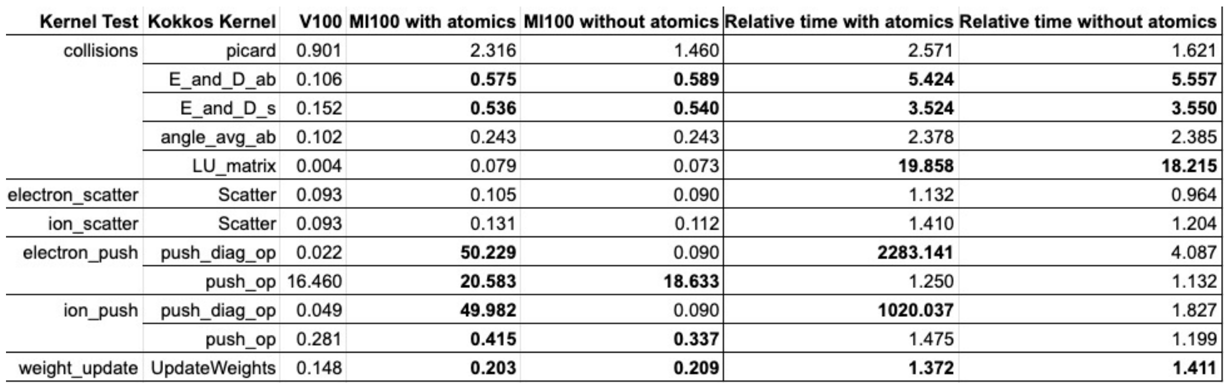
3. On the V100, performance was increased by exploiting shared memory. On the MI100 shared memory appears to have less benefit. We need help understanding why that is.

In Fig. 69 for Crusher we see nearly 24× speedup comparing our baseline version of SNAP to our latest version. Turns out that NVIDIA V100 was a very good proxy for AMD MI250X.

We currently only project about a 1.8× speedup for full Frontier vs full Summit, which is alarming based on the difference in FLOPS between the two machines. AMD is working on their compiler to better handle Kokkos abstractions, so we hope to see some more gains for Kokkos in general. But a large part of the issue with SNAP is the 6x smaller cache on MI250X vs V100. AMD profiled the L1 cache hit rate on MI250X and it was only 66% in the largest kernel in SNAP, vs about 90% on V100.

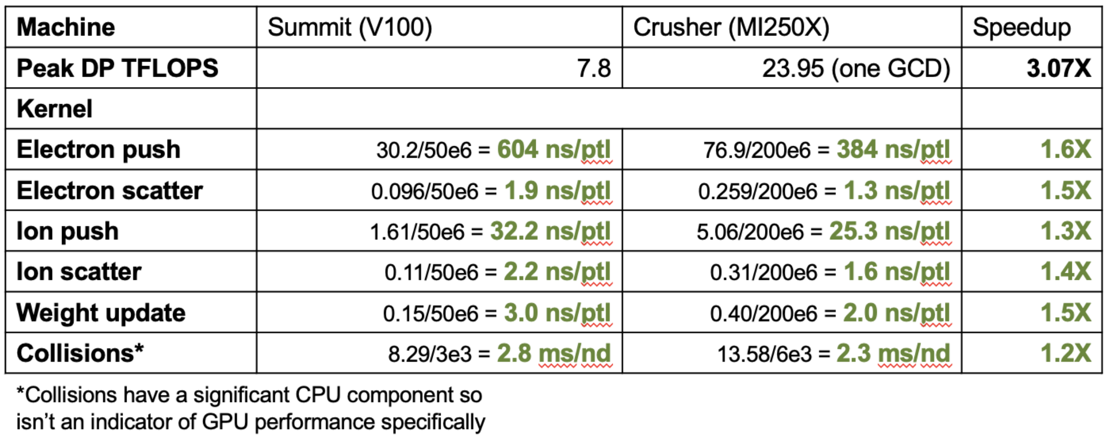
XGC

Poor performance of atomics has been confirmed as a contributor to overall poor performance on Spock. When atomics are not used (in which case, the results are incorrect but the code structure and computations



Kernel Test	Kokkos Kernel	V100	MI100 with atomics	MI100 without atomics	Relative time with atomics	Relative time without atomics
collisions	picard	0.901	2.316	1.460	2.571	1.621
	E_and_D_ab	0.106	**0.575**	**0.589**	**5.424**	**5.557**
	E_and_D_s	0.152	**0.536**	**0.540**	**3.524**	**3.550**
	angle_avg_ab	0.102	0.243	0.243	2.378	2.385
	LU_matrix	0.004	0.079	0.073	**19.858**	**18.215**
electron_scatter	Scatter	0.093	0.105	0.090	1.132	0.964
ion_scatter	Scatter	0.093	0.131	0.112	1.410	1.204
electron_push	push_diag_op	0.022	**50.229**	0.090	**2283.141**	4.087
	push_op	16.460	**20.583**	**18.633**	1.250	1.132
ion_push	push_diag_op	0.049	**49.982**	0.090	**1020.037**	1.827
	push_op	0.281	**0.415**	**0.337**	1.475	1.199
weight_update	UpdateWeights	0.148	**0.203**	**0.209**	**1.372**	**1.411**

Figure 70: Performance for XGC routines on Spock with and without atomics.



Machine	Summit (V100)	Crusher (MI250X)	Speedup
Peak DP TFLOPS	7.8	23.95 (one GCD)	**3.07X**
Kernel			
Electron push	30.2/50e6 = **604 ns/ptl**	76.9/200e6 = **384 ns/ptl**	**1.6X**
Electron scatter	0.096/50e6 = **1.9 ns/ptl**	0.259/200e6 = **1.3 ns/ptl**	**1.5X**
Ion push	1.61/50e6 = **32.2 ns/ptl**	5.06/200e6 = **25.3 ns/ptl**	**1.3X**
Ion scatter	0.11/50e6 = **2.2 ns/ptl**	0.31/200e6 = **1.6 ns/ptl**	**1.4X**
Weight update	0.15/50e6 = **3.0 ns/ptl**	0.40/200e6 = **2.0 ns/ptl**	**1.5X**
Collisions*	8.29/3e3 = **2.8 ms/nd**	13.58/6e3 = **2.3 ms/nd**	**1.2X**

*Collisions have a significant CPU component so isn't an indicator of GPU performance specifically

Figure 71: Performance for XGC kernels on Crusher and Summit.

are identical), XGC's major GPU kernels perform 1–5× slower on a MI100 than on a V100. The most expensive kernel on V100, the electron push, is 1.25× slower (atomics included) on MI100. Timing results are shown in Fig. 70.

In Fig. 71, on Crusher, individual XGC Kernels show a 1.2–1.6× performance improvement on AMD MI250X (one GCD) compared to a NVIDIA V100. This result is not unexpected. Further optimizations in the compiler, Kokkos, and XGC will improve these numbers over time. Performance issues on Spock are not present on Crusher, which outperforms the V100 particularly at high particle count.

In a full XGC production-like case, 64 Crusher nodes performed 2.3× slower than 256 Summit nodes. These configurations were chosen for comparison since they have similar theoretical peak FLOPS (12.0 PFLOPS for Summit, 12.3 PFLOPS for Crusher). In Fig. 72 Crusher took longer per time step than Summit, indicating that while there is already an acceleration per GPU, this acceleration falls short of 3.07×. This is consistent with the kernel performance tests.

The following issues were noted when running XGC on Crusher.

- Load imbalance appears to be worse on Crusher than on Summit. This may be an issue with our load balancing algorithm; it may also be an issue with out test case, which consists of only two time steps.
- CPU-GPU communication is taking more time on Crusher than on Summit. In addition to the interconnects being slower (36 GB/s per Crusher GCD vs 50 GB/s per Summit GPU), there are more particles per GPU. One solution is to implement the streaming `parallel_for`, hiding communication time; this feature has not yet been tested on Crusher.

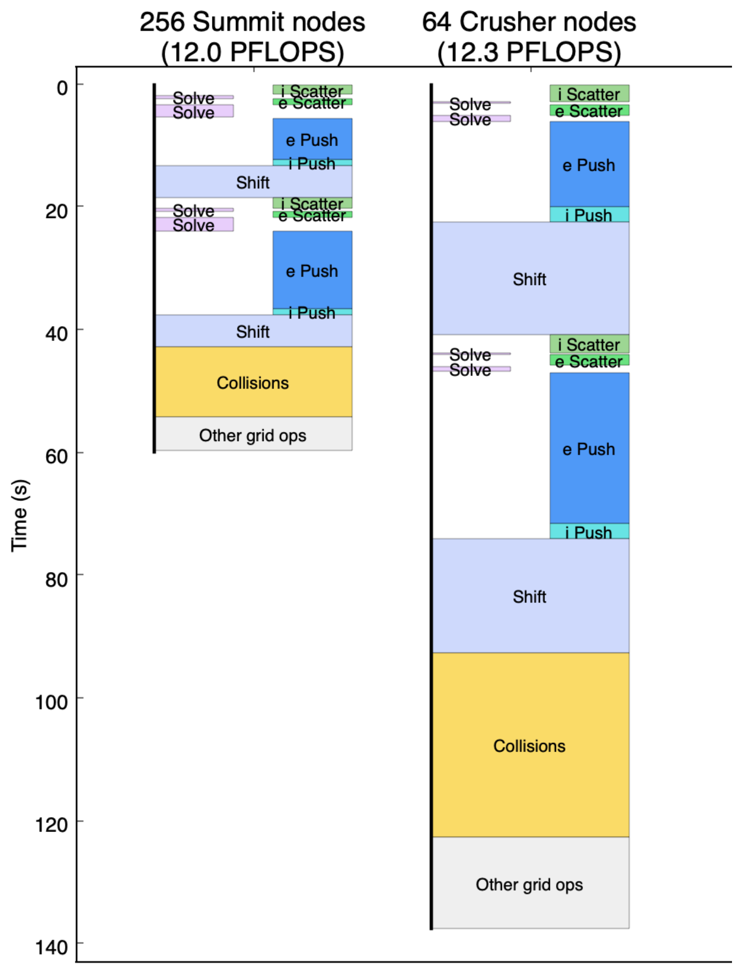


Figure 72: Performance for a full XGC production-like case on Crusher versus Summit.

Table 72: Currently planned AMReX integrations to reach the KPP-3 threshold.

Project	Sec	Code	Mesh	Particles	EB	Linear Solvers
WarpX	4.6	WarpX	Yes	Yes	Yes	Yes
MFIX-Exa	4.4	MFIX-Exa	Yes	Yes	Yes	Yes
Combustion	4.2	Pele suite[a]	Yes	Yes	Yes	Yes
ExaSky	5.2	Nyx	Yes	Yes	No	Yes
ExaStar	5.1	Castro	Yes	Yes	No	Yes
ExaStar	5.1	FLASH-X	Yes	Yes	No	No
ExaWind	4.1	AMR-Wind	Yes	No	No	Yes
ExaAM	3.5	TruchasPBF	No	No	No	Yes

[a]The Pele suite of codes consists of PeleC, PeleLM, and PelePhysics.

- The shift routine is disproportionately slower. This performance test does not use GPU-Aware MPI. An additional test is planned with this feature activated.
- The collisions routine inside of XGC appears to perform comparatively worse than the collisions kernel. This may be due to load imbalance. In any case, moving the collisions solver to the GPU should improve its performance by up to 40% according to performance tests with the Ginkgo library.
- GPU kernels in general are not reaching their ideal 3.07× speedup. Further profiling is necessary to understand why, and what solutions are available.

HACC

HACCabana has no software dependencies, other than Cabana and Kokkos. The team is in the process of developing a pure-Kokkos implementation to help identify a significant performance gap (∼100 ×) with natively written CUDA kernels. We compared single GPU runs obtained on Crusher (MI250X) to Summit (V100) and see similar timing (within ∼5 %) for our test problem. The unobtained speedup on MI250X is likely related to the performance gap we are investigating. No issues or blocking issues with compiling/running on Frontier/AMD early access systems (Spock, Crusher). No attempt to run on Spock/Crusher CPUs, but HACCabana has been run (very slowly) in serial mode on other CPUs.

7.3 AMReX

7.3.1 KPP Readiness

As a co-design center, AMReX falls under the KPP-3 metric, meaning that our success is determined by how many ECP application development projects successfully integrate AMReX into their application and successfully run on Frontier or Aurora. To pass, we need to demonstrate at least 7 such integrations, with a stretch goal of 14. The current set of application codes using AMReX is presented in Table 72. For each these codes, the AMReX Co-Design center earns one point towards its KPP-3 threshold if the code runs correctly on Frontier, and another if it runs correctly on Aurora. To provide more context for these applications we have also listed which aspects of AMReX are used by each of the codes.

For each of the above projects, AMReX will submit the following documentation:

1. Simulation output documenting that the application code uses AMReX on Frontier and/or Aurora.
2. A paper or similar documentation referring to the simulation code's use of AMReX.
3. A letter from the PI of the corresponding application development project affirming the use of AMReX in the simulation code.

Regarding item 1, the use of AMReX by an application code can be verified by examining the `stdout` from a simulation run. AMReX-based application codes must call `amrex::Initialize` prior to using any AMReX objects and `amrex::Finalize` at the end of the run to clean up any memory held in AMReX's `Arena` objects.

When these functions are called, information about the AMReX version used is printed to `stdout`. Thus, by examining the output of a simulation run, reviewers can verify the use of AMReX.

For example, documenting the integration of WarpX and AMReX will have three artifacts. We will first submit an output file from Frontier and Aurora analogous to the example shown in Code Output 7.1, which was collected from a run that was used to measure a preliminary FOM for a KPP run on 4263 Summit nodes.

Code Output 7.1: AMReX Output

```
Initializing CUDA...
CUDA initialized with 1 GPU per MPI rank; 25578 GPU(s) used in total
MPI initialized with 25578 MPI processes
MPI initialized with thread support level 3
AMReX (21.03-68-g92945ad3a356) initialized
WarpX (21.03-79-g106a228725cc)
PICSAR (b35f07243c51)
Level 0: dt = 1.9966466e-16 ; dx = 5.986394558e-08 ; dy = 5.986394558e-08 ; dz = 3.235553937e-07

Grids Summary:
  Level 0   409248 grids  858255261696 cells  100 % of domain
            smallest grid: 128 x 128 x 128  biggest grid: 128 x 128 x 128


STEP 1 starts ...

... <Simulation specific output goes here> ...

Total GPU global memory (MB) spread across MPI: [16128 ... 16128]
Free  GPU global memory (MB) spread across MPI: [6158 ... 10436]
[The         Arena] space (MB) allocated spread across MPI: [12096 ... 12096]
[The         Arena] space (MB) used      spread across MPI: [0 ... 0]
[The  Device Arena] space (MB) allocated spread across MPI: [8 ... 8]
[The  Device Arena] space (MB) used      spread across MPI: [0 ... 0]
[The Managed Arena] space (MB) allocated spread across MPI: [8 ... 8]
[The Managed Arena] space (MB) used      spread across MPI: [0 ... 0]
[The  Pinned Arena] space (MB) allocated spread across MPI: [221 ... 366]
[The  Pinned Arena] space (MB) used      spread across MPI: [0 ... 0]
AMReX (21.03-68-g92945ad3a356) finalized
```

We will also submit a reference to a paper describing the use of AMReX by WarpX, such as [42]. Finally, we will submit an email from the WarpX PI, J. L. Vay. A number of these emails from AD projects, including a letter from Vay, have already been received and will be included in the final KPP verification documentation.

7.3.2 Early Hardware Status

The Crusher machine, which has the same AMD MI200 GPUs as those that will be on Frontier, was not available as of the writing of this report. Thus, our results in this section are mostly from Spock and Arcticus. While the performance results obtained will likely differ from those we will get on the actual Exascale hardware, the process of running on the pre-Exascale hardware has been crucial for addressing software issues both in AMReX and in the supporting software stack.

To ensure that AMReX functions properly on the early hardware, we have regular CI and regression testing. Each pull-request to AMReX is subjected to compilation tests with the latest ROCm and oneAPI versions. Additionally, we have regular regression tests running on MI100, especially after a new ROCm release. For Intel, we have set up a nightly regression test suite running on an ATS machine provided by Intel Development Platform.

Currently, most AMReX functionality works on both MI100 and ATS hardware. Table 73 shows the status of the most recent regression test runs on recent NVIDIA and AMD GPUs. Figure 73 shows an example calculation using the "Tracer" test from the Table above. This example uses the core mesh and particle data structures in AMReX, as well as the AMReX I/O routines and the `AMRLevel` class interface for handling adaptive mesh refinement with and without subcycling in time. All of these features work as expected on the pre-exascale AMD and Intel hardware.

There are, however, a few known issues. The latest version of ROCm 4.5 has a bug that results in compiler error at link time if AMReX's EB-aware linear solver is enabled. We have reported the issue to the Frontier Center of Excellence, and HPE has filed a bug report to AMD.

Performance Microbenchmarks

Table 73: Current AMReX regression test suite status on pre-exascale hardware. The AMD results come from Tulip. All marked issues have been reported either to AMD.

Test	V100	MI100
CNS-RT	Pass	Pass
CNS-Sod	Pass	Pass
EB_Cell_Dir_2D	Pass	Link Fail*
EB_Cell_Dir_3D	Pass	Link Fail*
EB_Cell_Neu_2D	Pass	Link Fail*
EB_Cell_Neu_3D	Pass	Link Fail*
EB_MacProj_3D	Pass	Link Fail*
EB_Node_2D	Pass	Link Fail*
EB_Node_3D	Pass	Link Fail*
EB_Tensor_2D	Pass	Link Fail*
EB_Tensor_3D	Pass	Link Fail*
MLMG_ABecCom	Pass	Pass
MLMG_NodalPoisson	Pass	Pass
MLMG_PoisLev	Pass	Pass
ParticleMesh	Pass	Pass
ParticleReduce	Pass	Pass
ParticleTransformation	Pass	Pass
Redistribute	Pass	Pass
RedistributeBig	Pass	Pass
RedistributeMR	Pass	Pass
RedistributeMR_2D	Pass	Pass
RedistributeNonPeriodic	Pass	Pass
SortParticlesByCell	Pass	Pass
Tracers	Pass	Pass
Vector	Pass	Pass

*The EB-aware linear solver tests fail at link time with ROCm 4.5. However, they work in 4.3 and 4.4.

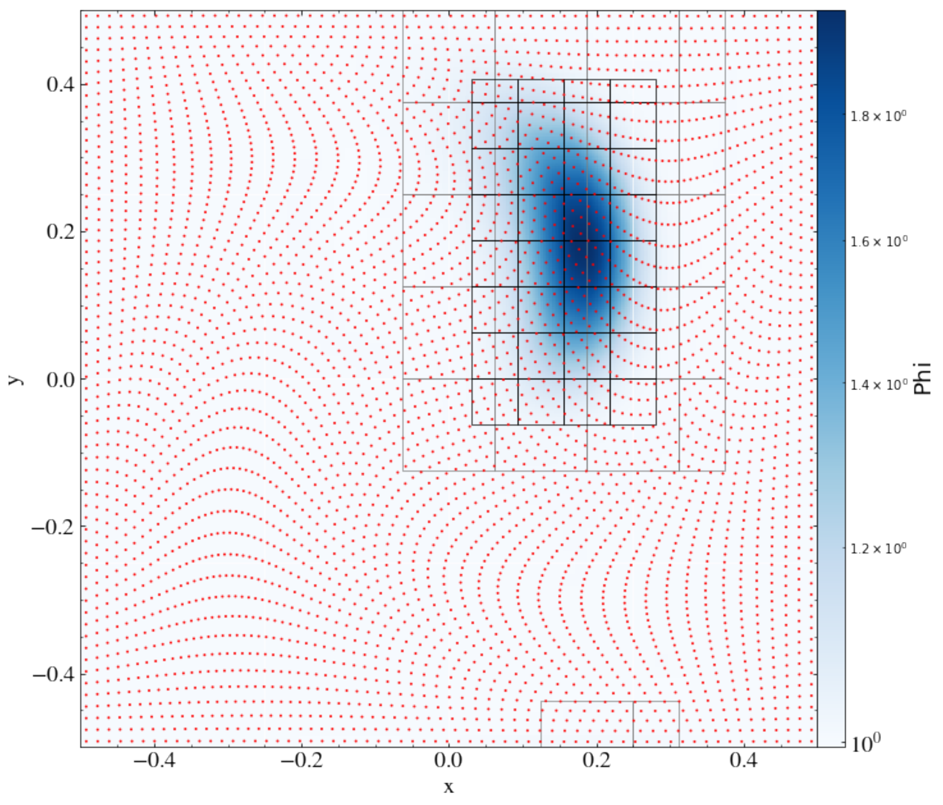


Figure 73: AMReX advection test problem with AMR and tracer particles. This example works on both ATS and AMD hardware.

Table 74: Performance results of microbenchmarks. The times (in seconds) on MI100, V100 and A100 are shown. ATS results are available available but not shown due to NDA restrictions.

Test	MI100	V100	A100
Memory bound	4.28e-4	5.53e-4	3.04e-4
Compute bound	9.85e-4	9.59e-4	7.03e-4
Branch	2.96e-4	3.55e-4	2.19e-4
Reduce sum	3.22e-4	2.81e-4	2.08e-4
Reduce max	3.27e-4	2.86e-4	2.13e-4
Scan	2.12e-3	1.35e-3	1.06e-3
Jacobi	9.61e-4	1.17e-3	5.54e-4
AoS	3.66e-3	4.17e-3	2.40e-3
GSRB	6.55e-4	3.85e-4	2.49e-4
Parser	2.74e-3	1.86e-3	1.11e-3

Table 75: Performance comparisons between various GPU hardware using WarpX's warm plasma benchmark. All runs used double precision.

GPU	Total time (s)	Deposition (s)	GatherAndPush (s)
MI100	116.4	67.5	40.2
MI200 (half)	37.3	21.1	7.4
MI200 (full)	20.5	11.0	4.0
V100	48.6	34.3	6.8
A100	33.1	24.1	4.1

To monitor the performance of common operations in AMReX, we have developed a set of performance microbenchmarks, including memory bound kernels, compute bound kernels, reductions, scan, smoothers in linear solvers, etc. These tests are run regularly on MI100 and ATS. Table 74 shows the latest results and the comparison to V100 and A100.

WarpX performance benchmark

It is also useful to consider the performance of more realistic compute kernels. Generally, complex compute kernels are not part of the AMReX framework; instead they live in associated application codes. However, these kernels operate on AMReX data structures and use the `amrex::ParallelFor` kernel launching mechanisms, thus their performance can tell us useful information about the operation of AMReX on various GPU platforms. We also note that this benchmark includes AMReX parallel communication functions such as ghost cell exchanges and particle redistribution; however, these operations take up a small fraction of the total run time on this test.

Table 75 shows the results of a standard "warm plasma" benchmark from the WarpX code using double precision. This test problem has been designed to particularly stress the particle-to-mesh (Deposition) and mesh-to-particle (GatherAndPush) interpolation operations, in that it uses 8 particles per cell and the relatively expensive Esirkepov deposition algorithm with 3rd order particle shapes. Thus, its performance depends critically on the performance of atomic operations on the underlying hardware, especially double-precision atomic adds. Since MI200 hardware is not currently available through Crusher, AMD ran this test on an internal cluster for comparison.

There are a few points to make about these numbers. First, the performance of the deposition operation was dramatically improved in going from MI100 to MI200 for double precision. The reason is that hardware support was added for double-precision atomic adds. Second, MI200 GPUs will have two dies per card, which AMReX treats as two independent GPUs. Thus, we have run two versions of the MI200 test, one using only 1 of the dies, and the other using two by assigning 2 MPI tasks to a single MI200. The scaling looks quite good, and using both of the dies, MI200 sees significantly better performance than either V100 or A100.

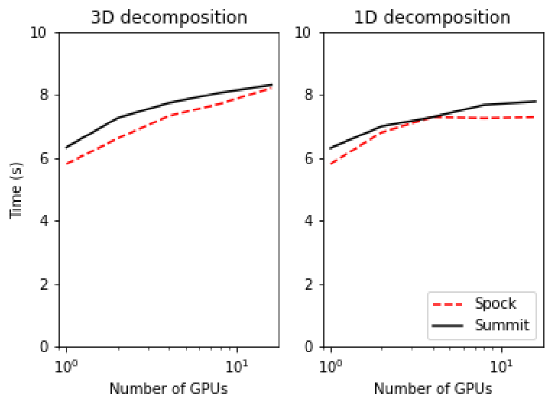


Figure 74: Performance results of the AMReX Redistribute benchmark, comparison between Spock and Summit.

Particle Redistribution Scaling

Particle redistribution—moving particles back to the proper grid, AMR level, and MPI task after their positions have changed—is a crucial operation in AMReX-based application codes that use particles, and it is a good stress test of several core operations on pre-Frontier hardware, including the GPU-aware parallel prefix sum, atomics performance, and the performance of the network and MPI implementation. Figure 74 shows the results of a weak scaling study using AMReX's standard particle redistribution benchmark. We have conducted two versions of this scaling study on up to 16 AMD MI100 GPUs on Spock, the largest number we have access to under the `ECP` queue. Both versions involve both "local" redistribute calls, in which particles are exchanged only between neighboring ranks, as well as "global" calls, in which particles can be exchanged between any pair of ranks in the entire MPI communicator. Additionally, both versions use two 128^3 grids with 1 particle per cell on each GPU.

The problem can be scaled up to more GPUs by doubling the domain size and doubling the number of GPUs used. We consider two strategies for increasing the problem size. The 3D domain decomposition adds boxes in all of the x, y, and z directions, whereas the "1D" domain decomposition only adds boxes in the z direction. The 3D version is harder to scale, because (for the "local" exchanges) the average number of communication partners per rank increases until you have $\approx 3^3 = 27$ GPUs, while the 1D version saturates at only $\approx 3^1 = 3$, where each rank only exchanges with the rank to its left and its right. Also shown is the corresponding problem run on the V100s on Summit, using only 4 of the GPUs per node for the sake of comparison. Overall, the performance on Spock on this benchmark is comparable, and in most cases slightly better, to that obtained on Summit, on which this benchmark has been shown to scale to almost the full machine [43]. Thus, with the caveat that this is not the final hardware that will be on Frontier, this benchmark did not reveal any red flags that would prevent AMReX-reliant codes from achieving their performance targets.

Linear Solvers

The performance of linear solvers is important for many AMReX applications (e.g., MFIX-Exa, Nyx, and PeleLM). Linear solvers contain a lot of operations for which it can be difficult to achieve good performance on GPU machines (e.g., communication, reduction, and small kernels at coarse multigrid levels). Thus, a test of AMReX's variable coefficient Poisson solver is one of our standard benchmarks. In a weak scaling study on Spock, and the times on 1 and 8 GPUs were 2.16 and 2.40 seconds, respectively. For comparison, the times on Summit were 1.96 and 2.11 seconds, again on 1 and 8 GPUs, respectively. The difference in time is primarily due to the red-black Gauss-Seidel smoother kernel; with one GPU, the times for just the smoother were 0.80 and 1.28 seconds on V100 and MI100, respectively. The performance difference is consistent with what we have observed with the GSRB microbenchmark in Table 74.

Table 76: CEED integrations and artifacts.

Integration	Platform targets	Expected artifacts
CEED↔MARBL (MFEM↔MARBL)	Pre-Exascale, Frontier, El Capitan	Current and Future ASC milestone reviews/reports, MARBL code release, repository tags, simulation outputs.
CEED↔ExaSMR (NekRS↔OpenMC) (NekRS↔Shift)	Pre-Exascale, Frontier, Aurora	Joint publications on state-of-the-art reactor thermal-hydraulics simulation. ECP milestone reports. Public NekRS release that is fully optimized for pre-exascale systems and exascale platforms.
CEED↔ExaAM (MFEM↔ExaConstit)	Pre-Exascale, Frontier	ExaAM yearly reports, releases of ExaConstit on its LLNL GitHub page, presentations of ExaConstit at conferences. Build scripts, input decks, and Caliper output files.
CEED↔ExaWind (NekRS↔AMR-Wind)	Pre-Exascale, Frontier, Community Standards	Bake-off study on exascale system and report on the performance and modeling V&V in collaborative publication. ECP Milestone reports. GABLS example in Nek5000/NekRS repository.
CEED↔ExaWind (libCEED↔Nalu-Wind)	Frontier	Boundary layer benchmark specification, performance comparison in ECP milestone reports.
CEED↔E3SM (libCEED↔E3SM)	Frontier, Community Standards	BP specification and extension to sphere, performance baseline in milestone report, continually tested libCEED-based examples reporting performance.

7.4 CEED

7.4.1 *KPP Readiness*

The CEED team plans to perform the KPP-3 application integrations shown in Table 76. Most of these integrations will be verified as part of the application's KPP-1/KPP-2 runs, but in some cases, we will need additional HPC resources for the verification. These have been requested in our KPP-3 table in ECP Confluence, e.g., NekRS↔OpenMC/Shift will require 180 k node hours on both Frontier and Aurora.

Expected artifacts for each integration target are listed in Table 76. Some examples of such artifacts include:

- CEED papers documenting integrations with MARBL, ExaSMR, ExaAM, and ExaWind:
  - *Efficient Exascale Discretizations: High Order Finite Element Methods*
  - *GPU Algorithms for Efficient Exascale Discretizations*
- The MARBL L1 milestone report and KPP-3 completion letter from the PI that document the CEED↔MARBL integration:
  - *The Multiphysics on Advanced Platforms Project*
  - MARBL code release, repository tags, simulation outputs are available at Lawrence Livermore National Laboratory (LLNL)
- Publications and INCITE award that document the CEED↔ExaSMR integration:
  - *Initial full core SMR simulations with NekRS*
  - *Feasibility of full-core pin resolved CFD simulations of small modular reactor with momentum sources*
  - 440 k node hours INCITE award: *Ab-initio Nuclear Structure and Nuclear Reactions*

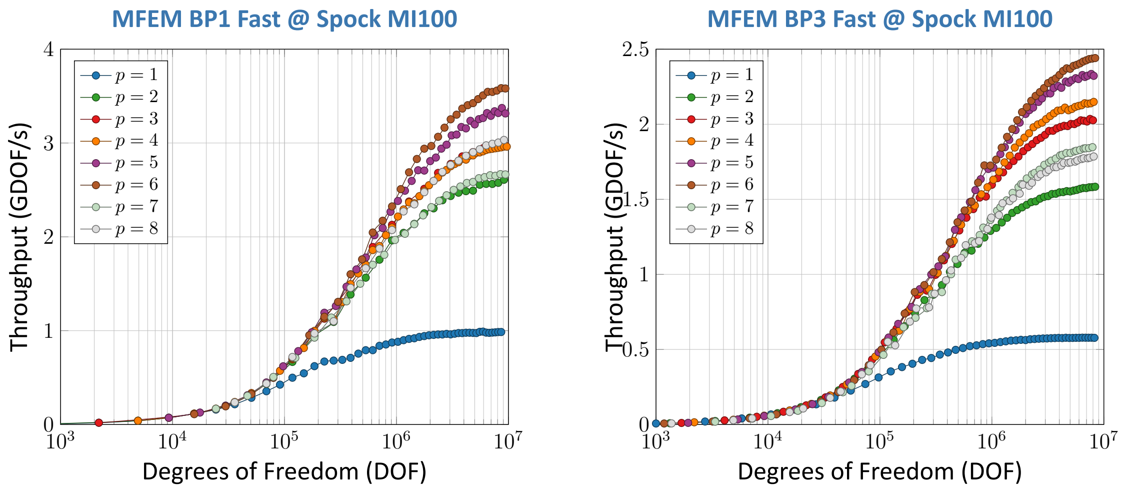


Figure 75: CEED BP1 and BP3 benchmark on a single AMD MI100 GPU for the new non-deterministic (fast) MFEM kernels.

- The ExaConstit repository (with MFEM related changes), presentations, and KPP-3 completion letter that documents the CEED↔ExaAM integration
  - `https://github.com/LLNL/ExaConstit`
  - See also the ExaConstit presentation at WCCM 2020: *How big do you need to go—application of large-scale crystal plasticity modelling to as-built AM IN625*
- Letters from ExaWind and ExaSMR documenting CEED integration activities.

7.4.2 *Early Hardware Status*

Spock status

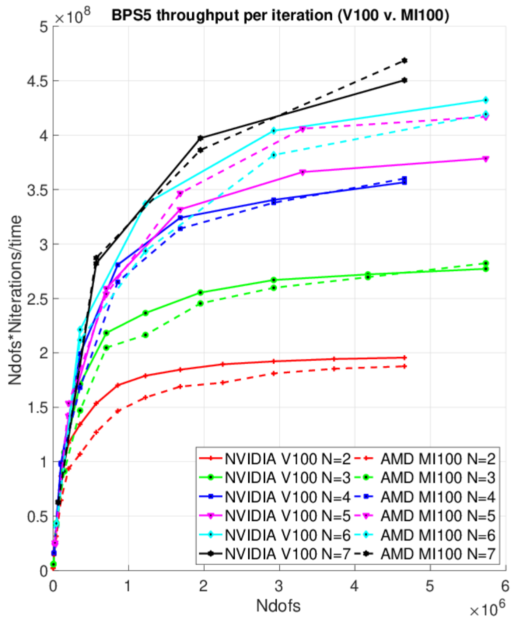
All CEED codes have been ported to AMD GPUs and have run on early-access Frontier systems. The results of these MI100 runs have been documented in Section 3 of the CEED-MS37 report. A selection of these results plus several recent updates are reported below.

MFEM fast Kernels on AMD MI100 GPUs

Figure 75 presents the performance of newly developed MFEM non-deterministic GPU kernels on a single AMD MI100 hosted on the Spock Frontier early-access system for the BP1 and BP3 CEED benchmarks. The overall performance is comparable to the one we see on NVIDIA V100 GPUs.

libParanumal Performance on AMD MI100 GPUs

The CEED/VT team performed detailed performance analysis of libParanumal and related CEED codes on the AMD MI100 GPUs hosted on the Spock early-access system. Overall, the libParanumal version of the CEED BPS5 preconditioned finite element elliptic solver benchmark running on the AMD MI100 with the OCCA HIP backend matches or exceeds the performance of an NVIDIA V100 via the OCCA CUDA backend for sufficiently large problem sizes. The measured performance is in line with our throughput model projections with the main performance limiter of the hottest kernels being the total aggregate available register capacity and local data share cache. On the other hand, the MI100 compensates for slower matrix-vector products with some performance gains for streaming kernels due to the enhanced memory throughput of that GPU compared to the V100. The performance of the MI100 for small problem sizes trails that of the V100 and this has been directly attributed to the longer launch latencies of kernels running on the MI100


(a)

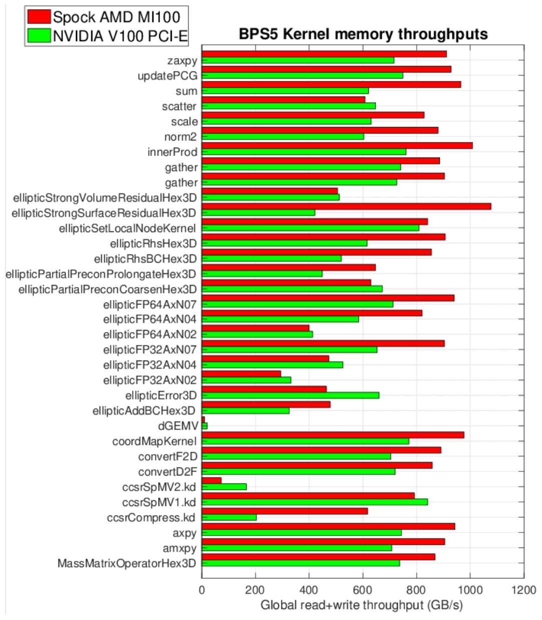

(b)

Figure 76: (a): comparing libParanumal CEED BPS5 throughput as a function of problem size for polynomial degrees 2 : 7 on an AMD MI100 on Spock and an NVIDIA V100 PCI-E GPU. (b): comparing individual kernel throughputs for 24^7 elements of degree 7 (4.6 M global degrees of freedom). The BPS5 elliptic solver includes a heterogeneous pMG-AMG based multigrid preconditioner employing mixed precision together with Chebyshev iterations accelerated with specialized smoothing operators.

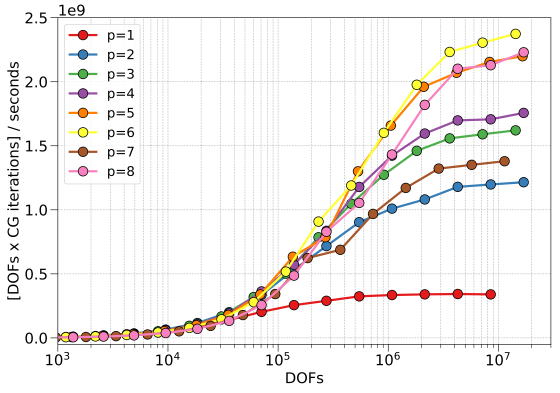
compared to the V100 despite our using the `AMD_DIRECT_DISPATCH` option. In the Fig. 76 we compare per kernel throughputs for the CEED BPS5 implementation on an AMD MI100 hosted on Spock and an NVIDIA V100 PCI-E model.

NekRS/ExaSMR 17 × 17 Rod-Bundle Strong Scaling Performance on Spock

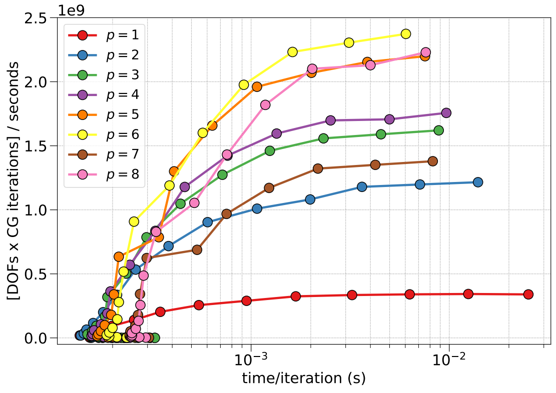
With a new release of NekRS 21.1, we performed strong-scaling studies of a 17 × 17 pin bundle. The mesh comprises 27,700 elements in the x-y plane and is extruded in the axial (z) direction with 3 layers. In this study, we do not use characteristics-based time stepping, but instead use a more conventional semi-implicit scheme that requires CFL 0.5 because of the explicit treatment of the nonlinear advection term. Table 77 presents strong scaling results on Spock for a case with $E = 83{,}100$ and $N = 7$ ($n = 28$ M) for n/P ranging from 5.7 M down to 1.7 M. We see 68% efficiency using $n/P = 2.3$ M on Spock (while on Summit we get 82% efficiency using $n/P = 2.0$ M).

Table 77: NekRS strong scaling performance on Spock (AMD MI100), 17 × 17 rod-bundle with $E = 83{,}100$, $n = 28$ M.

Node	GPU	n/P	v_i	p_i	t_{step}	Eff
2	5	5.7007×10^6	3	1	1.5523×10^{-1}	100
2	6	4.7506×10^6	3	1	1.3193×10^{-1}	98
2	8	3.5629×10^6	3	1	1.0567×10^{-1}	92
3	12	2.3753×10^6	3	1	9.5255×10^{-2}	68
4	16	1.7815×10^6	3	1	7.6610×10^{-2}	63



(a)



(b)

Figure 77: Performance of libCEED hip-gen backend inside MFEM for the BP3 (diffusion) benchmark. Results obtained with ROCm 4.3 on one node of Spock using one MI100 GPU.

BP3 for MFEM + libCEED hip-gen backend

The latest BP3 benchmark results for the libCEED hip-gen backend on one node of Spock are seen in Fig. 77. These results use the hip-gen backend inside MFEM. The y-axis shows the same metric in both plots, computing the DOF processing rate within the iterative solver; (a) has the problem size on the x-axis, while (b) considers time per iteration. The peak throughput achieved here is still lower than previous cuda-gen results, but the gap is much smaller than in previous results. However, the minimum time per iteration, seen in the clustered x-axis values of smaller problem sizes in the right plot, has not yet been meaningfully reduced.

Aurora status

Several backends have been developed in CEED software packages to target the Intel GPUs in Aurora, including a DPC++ backend in OCCA and a SYCL backend in MFEM. In addition, the NekRS code and Omega_h meshing library are being actively ported to the Aurora early-access systems at ALCF.

OCCA DPC++ Backend

OCCA v1.2.0includes a new DPC++ backend made possible by a joint project between ALCF, Intel, and Shell. The main highlights of the DPC++ backend are:

- Using freely available DPC++ implementations, this backend can be run on CPUs, GPUs, and FPGAs.
- Memory allocation/deallocation is handled using the DPC++ USM functions, which are now part of the SYCL 2020 standard.
- OKL kernels are translated to lambda functions for transparency and performance purposes.
- The OKL `@outer` and `@inner` loop indices are mapped to work-group IDs and work-items, respectively in a similar fashion to OpenCL.

During development the Intel oneAPI DPC++ compiler was used. Intel GPU hardware was used for testing, including the JLSE Aurora testbeds at ALCF. Soon testing will be expanded to include other DPC++ implementations and other vendor hardware.

The OCCA DPC++ backend has been successfully integrated into the primary OCCA repository on GitHub[6]. A feature on this work was also posted on the ACLF website[7].

MFEM SYCL backend

[6]https://github.com/libocca/occa/pull/494
[7]https://www.alcf.anl.gov/news/aurora-software-development-bringing-occa-open-source-library-exascale

A new SYCL backend was developed in MFEM and is currently available in a development branch at https://github.com/mfem/mfem/tree/sycl. We have tested the build of MFEM and the Laghos miniapp with Intel compilers: dpcpp and clang++ (`-fsycl`) and have run the CEED benchmark tests with MFEM on several Aurora early-access systems.

NekRS

NekRS has been successfully built on the JLSE Aurora testbed using the Aurora SDK stack, which includes the Intel LLVM C, C++, and Fortran compilers, as well as a version of MPICH optimized for Aurora hardware. Standard test cases from the NekRS examples were run on the Iris, Yarrow, and Arcticus nodes to verify accuracy for both the OCCA OpenCL and DPC++ backends. Additionally, the ExaSMR single-rod case has been run to obtain preliminary performance estimates. NekRS has been integrated into the Continuous Integration (CI) pipeline used by ALCF staff so that application correctness and performance can be tracked over time as the Aurora SDK receives updates from Intel.

While much of the performance data is still under embargo, ALCF staff continue to track and analyze NekRS performance as new hardware is available to ensure that application performance is as expected. Collaboration with the Intel MKL and MPI teams is also ongoing. Work on OCCA and NekRS integration efforts in collaboration with ALCF and Intel has been presented at SC20[8], SC21[9], and the oneAPI DevSummits at ISC21 and SC21[10].

Progress Porting Omega_h to Aurora Testbed Systems

The Omega_h GPU accelerated conforming mesh adaptation library supports execution on AMD GPUs via a HIP+RocThrust backend and on NVIDIA GPUs using either Kokkos+Thrust or CUDA +Thrust. Efforts are focused on porting the Omega_h backend to Kokkos using SYCL for execution on Intel GPUs in the Aurora testbed systems. The Kokkos SYCL backend is the feature-complete, long-term, backend for use on Intel GPUs and is publicly available on GitHub. Omega_h's use of this backend requires porting Thrust calls to oneAPI DPL or Kokkos. Thrust calls that cannot currently be ported into Kokkos, such as computing the permutation to sort an array of integers, will be implemented with DPL. Unit tests of the ported parallel reduction operations using user-defined types are passing as are `parallel_for` kernel launches. Porting of sorting operations to DPL is ongoing. This effort is being supported by the Kokkos SYCL backend developers. In the last period efforts were focused on running Omega_h on the JLSE testbed system. In the next period development will continue using the JLSE system and the newly available Intel DevCloud system that hosts Ponte Vecchio GPUs.

7.5 ExaGraph

7.5.1 KPP Readiness

ExaGraph KPP goals and status are listed in Tables 78 through 82. Table 78 supports NWChemEx (§ 3.2), Table 79 supports ExaBiome (§ 6.3), and Table 81 supports ExaWind (§ 4.1). Tables 80 and 82 supports the ST projects SuperLU/STRUMPACK and Trilinos, respectively.

7.5.2 Early Hardware Status

Graph Neighborhood Evaluations on Spock

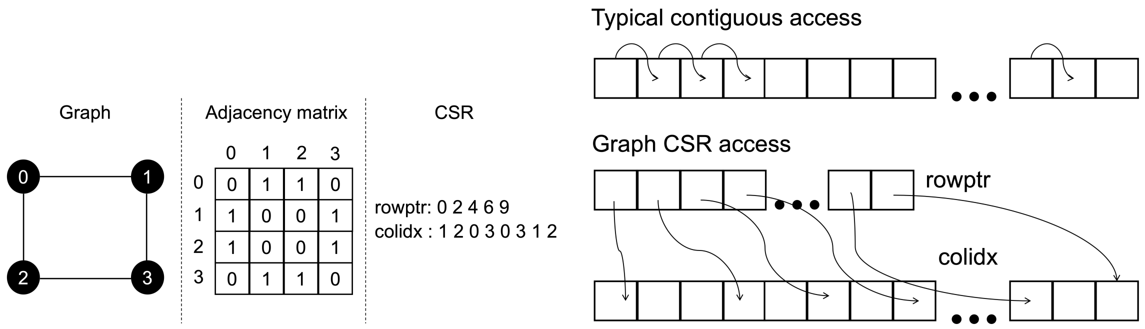
Graphs are usually represented through a variant of Compressed Sparse Row or Column (CSR/CSC) formats to store the nonzero elements. Considering the graph as a 2D adjacency matrix, the CSR format consists of two arrays to store the row pointers and column indices (optionally, a third array can also be used to store edge weights). Due to this representation, graph accesses can be highly irregular. Fig. 78 demonstrates a sample CSR format in comparison with a regular contiguous access pattern.

The *graph neighborhood* of a vertex v in a graph $G = (V, E)$ (where V and E represents the respective numbers of vertices and the corresponding edges) is the set of vertices including v and its neighbors, $u \in \text{adj}(v)$.

[8] https://sc20.supercomputing.org/presentation/?id=tut108&sess=sess242
[9] https://sc21.supercomputing.org/presentation/?id=tut105&sess=sess207
[10] https://www.oneapi.io/event-sessions

Table 78: ExaGraph KPP-3 Goal 1: Load balancing tasks in Self-Consistent Field (SCF) computations in computational chemistry.

Capability Description	Design and implement efficient Submodular *b*-Matching algorithms for task assignment in the compute-intensive Fock matrix construction. A *b*-Matching is a subgraph of the given input graph, where the degree of each vertex *v* is bounded by a given natural number *b*(*v*). In linear (or modular) *b*-Matching the objective function to be maximized is the sum of the weights of the edges in a *b*-Matching. In Submodular *b*-Matching the objective function is submodular (nonlinear, representing diminishing returns).
Integration Goals	The default load balancing used in NWChemEx is to assign iteration indices of the outermost two loops in the Fock matrix build across MPI ranks using an atomic counter based work sharing approach. All MPI ranks atomically increment a global shared counter to identify the loop iterations to execute. This approach limits scalability of the Fock build since the work and number of tasks across MPI ranks are not guaranteed to be balanced. A *b*-Matching based approach will provide efficient load balancing to scale NWChemEx to larger number of nodes on a supercomputer, as well as minimize the total runtime significantly.
Relevant P6 Activities	ADCD05-46
Target Environment	Frontier, Aurora, Pre-Exascale (Summit)
Verification	KPP-3. As a representative bio-molecular system we chose the Ubiquitin protein to test performance, varying the basis functions used in the computation to represent molecular orbitals, and to demonstrate the capability of our implementation to handle large problem sizes. The assignment algorithm is general enough to be applied to any scenario where such computational patterns exist, and does not depend on the molecule or the basis functions used.
Producer	ExaGraph teams at Purdue and PNNL
Consumer	NWChemEx team at PNNL
Backup Activity	A formulation using vertex-weighted matching algorithm and graph partitioning.
Artifacts	Open source software, demonstration of integration through peer-reviewed publications, and letter of support from appropriate NWChemEx leadership.



	0	1	2	3
0	0	1	1	0
1	1	0	0	1
2	1	0	0	1
3	0	1	1	0

Figure 78: Sample graph CSR and contrasting contiguous and graph (CSR) access pattern.

Table 79: ExaGraph KPP-3 Goal 2: ExaBiome.

Description	HipMCL and other ExaBiome codes require semi-ring sparse matrix-matrix multiplication (SpGEMM), which will be provided by ExaGraph. All known high-performance implementations of SpGEMM on GPUs run on the standard floating point algebra. However, combinatorial algorithms, such as those used in ExaBiome's long-read assembler and protein similarity search (PASTIS) tools, require SpGEMM to run on a different algebra, which in general is a semi-ring with user-defined addition and multiplication operators, as well as an additive identity.
Integration Goals	HipMCL and other ExaBiome codes such as ELBA (long-read assembler) require semi-ring (masked) sparse matrix multiplication, which will be provided by ExaGraph. This KPP is contingent on our project being extended to FY23 (at least the first three quarters). More info at https://jira.exascaleproject.org/browse/INT-1506.
Relevant P6 Activities	To be done in FY23. This activity and the KPP is contingent on our project's extension to FY23.
Target Environment	Frontier, Pre-Exascale (Perlmutter)
Verification	planned
Producer	ExaGraph team at LBL
Consumer	ExaBiome team at LBL
Backup Activity	planned
Artifacts	Open source software, demonstration of integration through peer-reviewed publications.

The graph neighborhood access pattern is demonstrated in Algorithm 1. The loops in Algorithm 1 are equivalent to traversing the graph CSR.

Algorithm 1 Graph neighborhood access pattern.
Input: $G = (V, E)$, (undirected) graph G.

```
1: for $v \in V$ do
2:     for $u \in \mathrm{adj}(v)$ do                ▷ Neighbors of $v$
3:         Perform some work with $\{v, u\}$
4:     end for
5: end for
```

Instead of bytes transferred per second to report bandwidth, we use the traversed-edges-per-second metric, introduced by the Graph500 benchmark. However, the volume of traversed edges in the graph neighborhood access pattern is quite different from the Graph500 breadth-first search (BFS): BFS traverses edges proportional to $|V|$ edges (from a source vertex), whereas the graph neighborhood pattern traverses edges proportional to $|E|$ edges. Proxy simulations of graph applications often need the actual graph, since the performance and quality of a graph application broadly depends on the structure of a graph. Thus, we provide option to load real-world graphs unlike synthetic benchmarks. To improve locality of graph neighborhood accesses on NUMA systems, we leverage the first-touch placement policy by parallelizing the graph loading method such that each thread owns roughly equivalent number of vertices and all the associated edges.

A Spock node consists of a single socket 64-core (128 threads) AMD EPYC 7662 "Rome" CPU on four quadrants (NUMA nodes), with 32 MB L2 and 256 MB L3 caches with 256 GB DDR4 memory (8 memory channels), and 4 AMD MI100 GPUs with 32 GB HBM2 memory/GPU.

On the CPU platforms, we use an OpenMP worksharing-loop construct to parallelize the outer loop over vertices (see Algorithm 1), using the default static schedule. For the OpenMP offload version, we create target

Table 80: ExaGraph KPP-3 Goal 3: SuperLU and STRUMPACK.

Description	SuperLU and STRUMPACK has symbolic factorization but the code has been unsustainably complex. The goal is to reimplement symbolic factorization in GraphBLAS for sustainability to new (exascale) architectures. We have been developing edge-based and path-based symbolic factorization implementations in GraphBLAS. We will port that implementation to GPUs via an extension of the GraphBLAST library, and to distributed-memory architectures via the distributed GraphBLAS we have been developing over the Combinatorial BLAS library.
Integration Goals	Our GraphBLAS-based symbolic factorization codes will be integrated to SuperLU and STRUMPACK. Our KPP-3 threshold for this integration has two components. One is to be able to do symbolic factorization of sparse matrices that do not fit into a single node of the cluster, and the other is to beat the performance of an existing single-node solution with distributed-memory versions. More info at https://jira.exascaleproject.org/browse/INT-1516.
Relevant P6 Activities	ADCD05-12
Target Environment	Frontier, Aurora, Pre-Exascale (Perlmutter)
Verification	KPP-3. For scalability, we will use the largest problems where SuperLU and STRUMPACK has been known to struggle with. For correctness testing, we will use standard problems SuperLU and STRUMPACK already runs on.
Producer	ExaGraph team at LBL
Consumer	SuperLU and STRUMPACK team at LBL
Backup Activity	planned
Artifacts	Open source software, demonstration of integration through peer-reviewed publications.

Table 81: ExaGraph KPP-3 Goal 4: ExaWind.

Description	We will provide enhanced partitioning capabilities for ExaWind, in particular, the MueLu solver.
Integration Goals	Test and evaluate new partitioning strategy (Sphynx or Quotient graph) in MueLu within ExaWind
Relevant P6 Activities	ADCD05-10, ADCD05-43
Target Environment	Pre-exascale
Verification	planned
Producer	Erik Boman, (Seher Acer)
Consumer	Luc Berger-Vergiat
Backup Activity	planned
Artifacts	Results from Summit

Table 82: ExaGraph KPP-3 Goal 5: Trilinos.

Description	We will integrate Sphynx into Trilinos/Zoltan2, which will provide an accelerator/GPU-based partitioner for a wide range of applications using Trilinos.
Integration Goals	Integrate Sphynx into the Zoltan2 package of Trilinos. This will enable applications that use Trilinos access to a GPU-enabled graph partitioner. ParMetis does not run on GPUs.
Relevant P6 Activities	https: ADCD05-10, ADCD05-43
Target Environment	Pre-exascale
Verification	planned
Producer	Erik Boman, (Seher Acer)
Consumer	Ichi Yamazaki, Jennifer Loe
Backup Activity	planned
Artifacts	Github PR/commit for Trilinos.

data regions to allocate device data and use the `target teams distribute` clause to distribute the iterations of the outer loop over vertices across the thread teams. Our sample `copy` kernel is shown in Code 7.1.

Code 7.1: ExaGraph Copy Kernel

```
for (int i = 0; i < nv; i++)
    for (int e = edge_indices[i]; e < edge_indices[i+1]; e++)
            edge_weights[e] = edge_list[e].weight;
```

We use the Cray programming environment and ROCm v3.0. We evaluate the kernels on three real world graphs, and the results are shown in Table 83. Structural differences of the graphs are captured in Table 84; $\mid E \mid_{max}$ denotes the maximum #edges associated with a vertex, $\widetilde{\mid E \mid}$ is the median #edges, and $\mid E \mid_{\sigma}$ is the standard deviation.

We observe that AMD MI100 GPU performance is up to 15× better than AMD CPU (on 128 threads), however the relative structure of the graph plays an important role in determining the performance. Graphs with comparable $\mid E \mid_{max}$ and $\widetilde{\mid E \mid}$ (on the small side) performs the best (like delaunay_n24), whereas graphs with maximum divergence between $\mid E \mid_{max}$ and $\widetilde{\mid E \mid}$ exhibits little or no performance improvement due to the load imbalance. While we use ROCm v4.3.0 (uses LLVM toolchain) on Spock, evaluations on the NVIDIA DGX platforms (at PNNL) uses LLVM 13.0 (with CUDA 10.1) on V100 GPU, whereas NVHPC v21.3 (with CUDA 11.1) on A100 GPU. Overall, we observed about 2–8× improvement for graphs with modest median to max #edges on NVIDIA V100/A100 GPUs relative to the AMD MI100 GPU.

The ultimate goal of the kernels is to guide the graph application developers in assessing the performance of this fundamental pattern on diverse input graphs. Our graph neighborhood access benchmark suite is open source and available on GitHub: https://github.com/sg0/neve/tree/dev.

Zoltan2 and Sphynx

The Zoltan2 package in Trilinos relies mainly on Kokkos for performance portability to different architectures. The Sphynx partitioner has been integrated into Zoltan2. We have successfully built Zoltan2 and Sphynx both on Summit (NVIDIA GPU) and on Spock (AMD GPU). The porting effort of certain Trilinos packages to AMD by the ECP Sake team has been helpful. By relying on Trilinos and Kokkos, only a small effort was needed to complete the initial porting. We have identified several design decisions in Zoltan2 that appear to be quite inefficient in hybrid CPU-GPU environments, and will revisit them.

Ripples

Table 83: Performance in GTEPS (Giga-TEPS, higher is better) of graph neighborhood access kernels on OLCF Spock platform vs. NVIDIA GPUs using three graphs. For Spock, we also include AMD "Rome" CPU results on 128 threads.

Graph kernels	OLCF Spock (ROCm)		NVIDIA DGX	
	AMD Rome (128)	AMD MI100	V100 (LLVM)	A100 (NVHPC)
	com-Orkut (| V |=3.07 M, | E |=234.37 M)			
Neighbor Copy	1.95	6.77	3.84	11.14
Neighbor Add	2.31	7.81	15.79	21.39
Neighbor Max	2.32	8.05	23.33	58.96
	delaunay_n24 (| V |=16.77 M, | E |=100.66 M)			
Neighbor Copy	1.65	20.43	11.93	23.10
Neighbor Add	1.99	29.90	30.35	64.32
Neighbor Max	1.99	31.93	30.92	69.23
	graph500-scale22-ef16(| V |=2.39 M, | E |=256.38 M)			
Neighbor Copy	1.68	1.92	3.71	3.7
Neighbor Add	2.23	1.92	4.68	3.81
Neighbor Max	2.28	2.51	8.30	15.20

Table 84: Structural differences of the graphs.

Graphs	$\|V\|$	$\|E\|$	$\|E\|_{max}$	$\|\widetilde{E}\|$	$\|E\|_{\sigma}$
com-Orkut	3.07 M	234.37 M	33 k	45	73.47
graph500-scale22-ef16	2.39 M	256.38 M	341 k	10	108
delaunay_n24	16.77 M	100.66 M	26	6	1.33

We have started the porting of Ripples (and cuRipples) to the HIP programming model. The first goal of the porting process is to provide with a functional implementation that will be further optimized in the next iterations. To maintain support for both NVIDIA GPU based machines (e.g., Summit) and AMD GPU based machines (e.g., Spock), we have developed a very thin decoupling layer that enables us to express the algorithm in a vendor-agnostic manner, but at the same time, provides the needed flexibility to optimize kernels for each device.

Ripples supports two diffusion models: Independent Cascade (IC) and Linear Threshold (LT). Currently, we have completed the porting of the IMM algorithm when using the LT model to HIP. We have found that one of our dependency, Tina's Random Number Generator (TRNG), needs some minor rework to support AMD GPUs. We are planning to work with upstream tasks to fix what is needed on our end. We are planning to start scalability studies on Spock in December.

NetAlign

Our initial porting efforts for Network Algorithm is based on OpenMP offloading to accelerators and performance comparisons with multi-threaded shared-memory implementations. Given two graphs, the NA algorithm finds the best one-to-one map of vertices in one graph to the vertices in the second graph. The reference shared-memory implementation (NetAlign) is an iterative algorithm where each iteration comprises of two phases: *(i)* A belief propagation (BP) based similarity score computation, and *(ii)* a maximum weight bipartite matching (MWM) with similarity scores as the edge weights. The BP computation comprises of a series of matrix-vector computation where memory accesses are structured, but the MWM computation has irregular memory access patterns. Due to the complex nature of the memory accesses and data movement between and within iterations, GPU implementation of NetAlign is challenging. We have completed the initial OpenMP offloading implementation of NetAlign with descriptive execution model strategy and have observed about 1.5$\times$ slowdown on Cori with NVC++ compiler. The slowdown increases for larger input problem sizes.

Our current efforts are along two independent strategies: *(i)* high performance implementations using OpenCL for computationally intensive kernels, and *(ii)* performance optimization of OpenMP-offloading based implementation targeting portability across GPUs from NVIDIA, Intel and AMD.

7.6 ExaLearn

The ExaLearn project is built around four pillars: Surrogates, Design, Control, and Inverse Solvers as well as cross-cuts, where they apply. Cross-cuts include I/O, Uncertainty Quantification (UQ), Scalability and Performance, and Anomaly Detection. The following section summarizes the status and progress in terms of these pillars.

7.6.1 KPP Readiness

Inverse Solvers

In the KPP-3 work for ExaAM (§ 3.5), the Inverse Solvers team is using the MEUMAPPS code (developed under the ExaAM project) for phase field simulations. The wall time of a single MEUMAPPS execution is of the order of 2–3 hours (often more) on about 6–7 Summit nodes. As such, the multiple executions that are required to tune the input parameters to control the nucleation process quickly becomes computationally prohibitive. The Inverse Solvers team is developing trained ML models that can predict the input parameters at which to execute the MEUMAPPS code in order to simulate dendritic microstructures with desired characteristics. As part of this effort, the main artifacts will be inverse solver software, training data sets of labelled microstructures and reduced order ML models to predict the input parameters for target 3D dendritic microstructures as inputs to the trained model(s). The ability to bypass expensive forward simulations in a highly resource-intensive tuning process will be the key advancement. In addition, peer-reviewed papers in international journals and conferences will continue to remain the main vehicles for dissemination of these advances to the domestic and global research community at large.

Control

We will provide the following deliverables through FY23: Scalable reinforcement learning (RL) framework (EXARL) open source code:

- Code for RL algorithms used to run our science problems, testing and verification working on at least one exascale platform.
- Scalable mechanisms to distribute and parallelize RL applications
- Test suite for checking the baseline framework functionality, quality of learning and scalability.
- Documentation for users and developers
- Verification that the included algorithms and applications are working properly at low and high scale.
- Scripts to run EXARL at scale
- Performance measurements of EXARL at scale
- Scripts to visualize performance and quality of learning.
- Spack package for the framework
- Tutorial slides
- RL application code, open source if possible:
  - Science simulations
  - Reward functions for science applications

Design

We plan two applications that both solve optimization problems using exascale computing:

- Electrolyte Design: The science objective is to identify molecules with suitable electrochemical properties, which we will approach by using machine learning to guide an ensemble of quantum chemistry computations. The code retrains machine learning models dynamically then uses Bayesian optimization techniques to reorder a task queue based on likelihood for producing meaningful data (e.g., data about molecules with favorable properties).
  - *Simulations*: We plan a set of runs of the design framework on Aurora where we progressively increase the number of nodes. In each case, we will measure the optimization performance (e.g., number of suitable molecules found over time) and the system utilization (e.g., utilization of nodes).
  - *Artifacts*: We will release the source code used for these campaigns along with documentation so that they can be replicated and repurposed by other teams. We also will work with our science partners in the Joint Center for Energy Storage Research (JCESR) to validate the predictions from our team.
  - *Minimum Requirements*: We will compare the science performance of the application by comparing it to a baseline search without dynamic retraining of machine learning models and verify our system does not experience a significant decline in system utilization as we increase to EFlops levels of utilization.
- Water Cluster Optimization: The science objective is to identify configurations of water that are energetically stable using a combination of reinforcement learning and quantum chemistry simulations. Similarly to our electrolyte design problem, the application should achieve better search performance by continually learning better strategies for assembling water clusters as more simulations are performed.
  - *Simulations*: As above, we target validating our application with progressively larger runs on Frontier or Aurora. Increasing scales will feature more learning and simulation workers operating in parallel, and we will verify the rate at which we evaluate new clusters increases accordingly.

- *Artifacts*: We will release the source code for our application as a demonstration of coupling reinforcement learning to simulation on Exascale systems. We will also establish a series of benchmark challenges using the results of our runs.
- *Minimum Requirements*: We will ensure that the increase in scale for our algorithm results in a greater diversity of water clusters being evaluated over a given time. Performance metrics could include measurements of the number of distinct clusters assessed per second and the entropy of the distribution of proposed cluster bonding networks.

Surrogates

We will generate Super Resolution (SR) simulations for the ExaSky project (§ 5.2). These will be trained on a set of low (LR) and high resolution (HR) Nyx simulations spanning an order of magnitude. The goal is to be able to generate SR surrogate simulations which step from the LR to the HR obviating the need to run the HR simulation. This will reduce the computational time required to handle HR Nyx simulations by more than an order of magnitude, approaching a 50–100× speed up. The SR surrogates will have to be physically viable with tight constraints on their systematics and potential biases.

We will run both Nyx simulations (at LR and HR) as well as train our SR surrogates on these machines. We will generate the full 3D sims along with the accompanying statistics to verify their accuracy using probability distribution functions (PDFs), two and three-point correlation functions, pixel histograms, and other statistical analyses. By comparing the SR and HR sims directly we can quantify that we have met the requirements of accurately reproducing the simulations.

I/O

We will use Aurora and Summit for our current tasks. We will be using these resources to calculate the Poincare surfaces of the magnetic fields within the simulation (Summit for now, Aurora later) and doing the data reduction on both Summit and Aurora. This will involve running WDMApp codes (§ 4.5) with Poincare surface generation turned on, and then with a standalone application written in VTK-m. We will use the WDMApp codes as a baseline and validate against these results. We will run the standalone application on a separate set of resources and use ADIOS to couple this to the simulation. We will use MGARD to reduce the data to acceptable tolerances to reduce the I/O load on the simulation. The following artifacts and analyses will be produced:

- WDMApp codes, VTK-m Poincare service;
- Poincare surface plots with raw and reduced data. We will also use ML to identify features in the Poincare surface plot;
- Features are preserved with the use of data reduction to lower the costs of I/O.

Anomaly Detection

We are developing an unsupervised anomaly detection algorithm, using higher-order joint moment tensors, to enable in-situ detection of ignition events in combustion simulations performed by the Combustion-Pele (§ 4.2) project. The KPP-2 target problem of Pele involves a simulation of a dual fuel multi-pulse combustion scenario in a closed chamber that represents a piston bowl configuration. The simulation involves complex chemical processes of two distinct types of fuels, injected separately into a turbulent environment, mimicking reactivity controlled ignition in IC engines. To facilitate targeted analyses in this simulation e.g., probing the thermochemical state, providing training data for constructing combustion surrogates, we will enable detecting the spatial-temporal regions where ignition occurs at inception. Our software is implemented as a stand-alone library, and the anomaly detection kernels will be deployed in Pele via the Ascent in-situ framework of the ALPINE project. This whole effort is an integration effort spanning Pele, ALPINE, and ExaLearn. The following artifacts and analyses will be produced:

- The artifacts will include code repository information, including commit hashes, for the GenTen library that contains the joint moment tensor kernel for anomaly detection. These will be combined with compatible repository and commit hashes of Ascent library, together with the details of the Pele KPP-2 target problem (problem set up, input deck, etc.).

- Completion will be marked by successful execution of the Pele KPP-2 target simulation with in-situ ignition detection performed using the GenTen kernel (deployed through Ascent). We will document the computational overhead of the anomaly detection kernel, which should be a small fraction ($< 10\%$) of the average time per time step taken by Pele.

Scalability and Performance

We will run the CosmoGAN model with and without the multi-GAN training technique on Frontier to demonstrate the scalable deep learning training for the ExaSky application. Additionally, we will demonstrate the use of the subgraph parallelism for transformer models on both BERT and the Pilot 2 BERT transformer model used by the CANDLE project. The following artifacts and analyses will be produced:

- We will include performance logs and possibly publications showing the scalability of these training techniques.
- These demonstrations will illustrate the engagement between ExaLearn and ExaSky and CANDLE respectively.

7.6.2 *Early Hardware Status*

Inverse Solvers

Efforts in the inverse applications thrust have not yet exercised any early hardware systems. However, we anticipate running/testing the performance of the parallel training infrastructure for inverse models, currently under development as part of FY22 effort, on the early hardware systems.

Control

We have been able to run our ExaBooster (accelerator control) application using EXARL on Perlmutter. Early investigations demonstrate a 2× speedup versus Cori GPU. We will run on Spock shortly. We currently do not have access to Arcticus or Crusher, but have requested access for Arcticus. Design: We have yet to evaluate our applications on early hardware. Work has instead focused on scaling on pre-exascale systems.

Surrogates

None to date due to the inability to run the versions of pyTorch, LBANN, and/or Keras we need to use these machines. We imagine that this will soon be remedied.

I/O

VTK-m is being tested and benchmarked on Spock and will move to Crusher soon. This is in collaboration with the VTK-m ECP project. The Poincare surface service has not been benchmarked, but will be when ready.

UQ

GenTen has been run extensively on NVIDIA GPU platforms such as Summit. In addition, the GenTen library is built on top of the Kokkos performance portability layer. Our path to porting to early hardware is primarily relying on the readiness of Kokkos, which is already running on the early access Frontier and Aurora systems. Our team, has early access to Spock, on which Kokkos has been deployed by other projects, and are close to testing out the compilation and measuring performance of GenTen kernels on this architecture. We expect to move to Crusher as soon as we get access and Arcticus when time permits.

Scalability and Performance

The LBANN scalable deep learning toolkit is running on both the HIP/ROCm software stack for the HPE/AMD systems and the oneDNN software stack for Intel systems. Our performance analysis has been primarily focusing on tuning the toolkit for the HIP/ROCm software stack and we are in on-going co-design efforts with the vendors to help identify and fix performance and correctness issues within their toolchain. Current results still leave significant room for performance optimization, specifically, the single

GPU performance of most of AMD's deep learning software libraries are significantly lagging behind NVIDIA's DL software stack. We are seeing performance results where an MI100 is still outperformed by a P100 GPU. There are also ongoing issue between the behavior of the single GPU performance in ROCm and the multi-GPU performance of the RCCL communication library. Furthermore, there are still capability gaps between the AMD DL stack and the NVIDIA DL stack, specifically, the current lack of an equivalent to CUDA Graphs is an issue, although that is expected to be remediated by early CY2022. Note that many of these performance gaps are also observed within the PyTorch DL toolkit.

Finally, we have encountered multiple issues with the software development toolchain over the past year. Examples of these include driver bugs that prevented GPU stream-aware communication in our Aluminum library, issues with MIOpen's SQLite embedded library, and the leaking of open file descriptors.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

REFERENCES

[1] *Summit*, 2020. [Online]. Available: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit (visited on 2020).

[2] *Frontier COE Workshop*, Sep. 2020. [Online]. Available: https://www.olcf.ornl.gov/frontier-coe-workshop.

[3] *Spock Quick-Start Guide*, 2021. [Online]. Available: https://docs.olcf.ornl.gov/systems/spock_quick_start_guide.html.

[4] *Crusher Quick-Start Guide*, 2021. [Online]. Available: https://docs-internal.apps.granite.ccs.ornl.gov/systems/crusher_quick_start_guide.html.

[5] *JLSE Testbeds*, 2021. [Online]. Available: https://www.jlse.anl.gov/hardware-under-development.

[6] A. Siegel, E. Draeger, J. Deslippe, T. Evans, M. Francois, T. Germann, D. Martin, and W. Hart, "Map applications to target exascale architecture with machine-specific performance analysis, including challenges and projections," Exascale Computing Project, Tech. Rep. ECP-U-AD-RPT_2021_00208, Mar. 2021. [Online]. Available: https://www.exascaleproject.org/reports.

[7] C. DeTar *et al.*, *MILC Collaboration code for lattice QCD calculations.* [Online]. Available: https://github.com/milc-qcd/milc_qcd.

[8] RBC-UKQCD Collaboration, *The Columbia Physics System.* [Online]. Available: https://github.com/RBC-UKQCD/CPS.

[9] B. Joo, R. Edwards, F. Winter, *et al.*, *The Chroma Software System for Lattice QCD.* [Online]. Available: https://github.com/JeffersonLab/chroma.

[10] K. Clark *et al.*, *QUDA library for lattice-QCD calculations on GPUs.* [Online]. Available: https://github.com/lattice/quda.

[11] P. Boyle *et al.*, *Data parallel C++ mathematical object library.* [Online]. Available: https://github.com/paboyle/Grid.

[12] D. Williams-Young, A. Bagusetty, W. de Jong, D. Doerfler, H. van Dam, A. Vazquez-Mayagoitia, T. Windus, and C. Yang, "Achieving performance portability in Gaussian basis set density functional tehory on accelerator based architectures," *Parallel Computing*, vol. 108, p. 102 829, 2021.

[13] *AMB2018-01 description*, 2020. [Online]. Available: http://www.nist.gov/ambench/amb2018-01-description.

[14] A. Sharma, S. Ananthan, J. Sitaraman, S. Thomas, and M. A. Sprague, "Overset meshes for incompressible flows: On preserving accuracy of underlying discretizations," *Journal of Computational Physics*, vol. 428, p. 109 987, 2021.

[15] P. Mullowney, R. Li, S. Thomas, S. Ananthan, A. Sharma, A. Williams, J. Rood, and M. A. Sprague, "Preparing an incompressible-flow fluid dynamics code for exascale-class wind energy simulations," in *Proceedings of the ACM/IEEE Supercomputing 2021 Conference*, ACM, 2021.

[16] *Explicit resource files*, https://www.olcf.ornl.gov/wp-content/uploads/2019/12/ERF.pdf.

[17] *The Extreme-scale Scientific Software Stack*, 2021. [Online]. Available: https://e4s-project.github.io.

[18] T. Esirkepov, "Exact Charge Conservation Scheme For Particle-In-Cell Simulation With An Arbitrary Form-Factor," *Computer Physics Communications*, vol. 135, no. 2, pp. 144–153, Apr. 2001, ISSN: 0010-4655.

[19] J. Cole, "A High-Accuracy Realization Of The Yee Algorithm Using Non-Standard Finite Differences," *Ieee Transactions On Microwave Theory And Techniques*, vol. 45, no. 6, pp. 991–996, Jun. 1997, ISSN: 0018-9480.

[20] M. Karkkainen, E. Gjonaj, T. Lau, and T. Weiland, "Low-Dispersionwake Field Calculation Tools," in *Proc. Of International Computational Accelerator Physics Conference*, Chamonix, France, 2006, pp. 35–40.

[21] B. M. Cowan, D. L. Bruhwiler, J. R. Cary, E. Cormier-Michel, and C. G. R. Geddes, "Generalized algorithm for control of numerical dispersion in explicit time-domain electromagnetic simulations," *Physical Review Special Topics-Accelerators And Beams*, vol. 16, no. 4, Apr. 2013, ISSN: 1098-4402. DOI: 10.1103/PhysRevSTAB.16.041303.

[22] J.-L. Vay, I. Haber, and B. B. Godfrey, "A domain decomposition method for pseudo-spectral electromagnetic simulations of plasmas," *Journal of Computational Physics*, vol. 243, pp. 260–268, Jun. 2013, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2013.03.010.

[23] T. Sukhbold, S. E. Woosley, and A. Heger, "A high-resolution study of presupernova core structure," *The Astrophysical Journal*, vol. 860, no. 2, p. 93, Jun. 2018, ISSN: 1538-4357. DOI: 10.3847/1538-4357/aac2da. [Online]. Available: http://dx.doi.org/10.3847/1538-4357/aac2da.

[24] A. Menon and A. Heger, "The quest for blue supergiants: Binary merger models for the evolution of the progenitor of SN 1987A," *Monthly Notices of the Royal Astronomical Society*, vol. 469, no. 4, pp. 4649–4664, Apr. 2017, ISSN: 1365-2966. DOI: 10.1093/mnras/stx818.

[25] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukic, S. Sehrish, and W. K. Liao, "HACC: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016. DOI: 10.1016/j.newast.2015.06.003.

[26] J. D. Emberson, N. Frontiere, S. Habib, K. Heitmann, P. Larsen, H. Finkel, and A. Pope, "The Borg Cube simulation: Cosmological hydrodynamics with CRK-SPH," *The Astrophysical Journal*, vol. 877, p. 85, 2019. DOI: 10.3847/1538-4357/ab1b31.

[27] A. S. Almgren, J. B. Bell, M. J. Lijewski, Z. Lukic, and E. V. Andel, "Nyx: A massively parallel AMR code for computational cosmology," *The Astrophysical Journal*, vol. 765, no. 1, p. 39, 2013. DOI: 10.1088/0004-637X/765/1/39.

[28] J. Sexton, Z. Lukic, A. Almgren, C. Daley, B. Friesen, A. Myers, and W. Zhang, "Nyx: A massively parallel amr code for computational cosmology," *Journal of Open Source Software*, vol. 6, no. 63, p. 3068, 2021. DOI: 10.21105/joss.03068. [Online]. Available: https://doi.org/10.21105/joss.03068.

[29] N. Frontiere, C. D. Raskin, and J. M. Owen, "CRKSPH—A conservative reproducing kernel smoothed particle hydrodynamics scheme," *Journal of Computational Physics*, vol. 332, no. 1, pp. 160–209, 2017. DOI: 10.1016/j.jcp.2016.12.004.

[30] N. Frontiere, K. Heitmann, E. Rangel, P. Larsen, A. Pope, I. Sultan, T. Uram, S. Habib, S. Rizzi, and J. Insley, *Farpoint: A high-resolution cosmology simulation at the Gpc scale*, submitted, 2021. arXiv: 2109.01956.

[31] K. Heitmann, N. Frontiere, E. Rangel, P. Larsen, A. Pope, I. Sultan, T. Uram, S. Habib, H. Finkel, and D. Korytov, "The last journey. I. An extreme-scale simulation on the Mira supercomputer," *The Astrophysical Journal Supplement Series*, vol. 252, no. 2, p. 19, 2021. DOI: 10.3847/1538-4365/abcc67.

[32] K. Heitmann, E. Lawrence, J. Kwan, S. Habib, and D. Higdon, "THE COYOTE UNIVERSE EXTENDED: PRECISION EMULATION OF THE MATTER POWER SPECTRUM," *The Astrophysical Journal*, vol. 780, no. 1, p. 111, 2014. DOI: 10.1088/0004-637X/780/1/111.

[33] S. D. Walsh, H. E. Mason, W. L. D. Frane, and S. A. Carroll, "Experimental calibration of a numerical model describing the alteration of cement/caprock interfaces by carbonated brine," *International Journal of Greenhouse Gas Control*, vol. 20, pp. 176–188, 2014), doi = doi:10.1016/j.ijggc.2014.01.004.

[34] Q. Li, C. Steefel, and Y.-S. Jun, "Incorporating nanoscale effects into a continuum-scale reactive transport model for co2-deteriorated cement," *Environmental Science & Technology*, vol. 51, no. 18, pp. 10 861–10 871, 2017. DOI: https://doi.org/10.1021/acs.est.7b00594.

[35] J. M. Wozniak, H. Yoo, J. Mohd-Yusof, B. Nicolae, N. Collier, J. Ozik, T. Brettin, and R. Stevens, "High-bypass learning: Automated detection of tumor cells that significantly impact drug response," in *2020 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC) and Workshop on Artificial Intelligence and Machine Learning for Scientific Applications (AI4S)*, 2020, pp. 1–10. DOI: 10.1109/MLHPCAI4S51975.2020.00012.

[36] K. Mehta, B. Allen, M. Wolf, J. Logan, E. Suchyta, J. Choi, K. Takahashi, I. Yakushin, T. Munson, I. Foster, *et al.*, "A codesign framework for online data analysis and reduction," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, IEEE, 2019, pp. 11–20.

[37] K. Mehta, B. Allen, M. Wolf, J. Logan, E. Suchyta, S. Singhal, J. Y. Choi, K. Takahashi, K. Huck, I. Yakushin, A. Sussman, T. Munson, I. Foster, and S. Klasky, "A codesign framework for online data analysis and reduction," *Concurrency and Computation: Practice and Experience*, Aug. 2021. DOI: 10.1002/cpe.6519. [Online]. Available: https://www.osti.gov/biblio/1817542.

[38] E. Suchyta, S. Klasky, N. Podhorszki, M. Wolf, A. Adesoji, C. Chang, J. Choi, P. E. Davis, J. Dominski, S. Ethier, *et al.*, "The Exascale Framework for High Fidelity coupled Simulations (EFFIS): Enabling whole device modeling in fusion science," *The International Journal of High Performance Computing Applications*, p. 10 943 420 211 019 119, 2021.

[39] J. Dominski, J. Cheng, G. Merlo, V. Carey, R. Hager, L. Ricketson, J. Choi, S. Ethier, K. Germaschewski, S. Ku, *et al.*, "Spatial coupling of gyrokinetic simulations, a generalized scheme based on first-principles," *Physics of Plasmas*, vol. 28, no. 2, p. 022 301, 2021.

[40] J. Y. Choi, J. Logan, K. Mehta, E. Suchyta, W. Godoy, N. Thompson, L. Wan, J. Chen, N. Podhorszki, M. Wolf, *et al.*, "A co-design study of fusion whole device modeling using code coupling," in *2019 IEEE/ACM 5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5)*, IEEE, 2019, pp. 35–41.

[41] X. Yu, S. Di, A. M. Gok, D. Tao, and F. Cappello, "cuZ-Checker: A GPU-based ultra-fast assessment system for lossy compressions," in *IEEE International Conference on Cluster Computing*, IEEE, 2021, pp. 307–319.

[42] A. Myers, A. Almgren, L. Amorim, J. Bell, L. Fedeli, L. Ge, K. Gott, D. Grote, M. Hogan, A. Huebl, R. Jambunathan, R. Lehe, C. Ng, M. Rowan, O. Shapoval, M. Thevenet, J.-L. Vay, H. Vincenti, E. Yang, N. Zaim, W. Zhang, Y. Zhao, and E. Zoni, "Porting warpx to gpu-accelerated platforms," *Parallel Computing*, vol. 108, p. 102 833, 2021, ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2021.102833. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819121000818.

[43] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. P. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, and M. Zingale, "AMReX: A framework for block-structured adaptive mesh refinement," *Journal of Open Source Software*, vol. 4, no. 37, p. 1370, 2019. DOI: 10.21105/joss.01370. [Online]. Available: https://ccse.lbl.gov/AMReX.