ECP-RPT-ST-0002-2020–Public

ECP Software Technology Capability Assessment Report–Public

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

November 19, 2020

ECP-RPT-ST-0002-2020–Public

ECP Software Technology Capability Assessment Report–Public

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

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

November 19, 2020

REVISION LOG

Version	Date	Description
1.0	July 1, 2018	*ECP ST Capability Assessment Report*
1.5	February 1, 2019	*Second release*
2.0	February 1, 2020	*Third release*
2.5	November 19, 2020	*Fourth release*

EXECUTIVE SUMMARY

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

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

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

- We highlight the progress with the Extreme-scale Scientific Software Stack (E4S) efforts. In particular, we discuss how E4S has emerged as a new first-class entity in the HPC ecosystem, enabling new conversations with users, facilities, vendors, other US agencies and international partners. We also highlight the development of the E4S DocPortal and Version 1.0 of E4S community policies, and E4S Spack build caches.

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

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

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

The Exascale Computing Project Software Technology (ECP ST) focus area represents the key bridge between Exascale systems and the scientists developing applications that will run on those platforms. ECP ST efforts contribute to approximately 70 software products (Section 2.1.3) in six technical areas (Table 1). Since the publishing of CAR V2.0, we continue to evolve the product dictionary of official product names, which enables more rigorous mapping of ECP ST deliverables to stakeholders (Section 2.1.4).

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

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

Mathematical Libraries: High-performance scalable math libraries have enabled parallel execution of many applications for decades. ECP ST is providing the next generation of these libraries to address needs for latency hiding, improved vectorization, threading and strong scaling. In addition, we are addressing new demands for system-wide scalability including improved support for coupled systems and ensemble calculations. See Section 4.3. The math libraries teams are also spearheading the software development kit (SDK) initiative that is a pillar of the ECP ST software delivery strategy (Section 2.1.2).

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

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

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

ECP ST Software Delivery mechanisms: ECP ST delivers software capabilities to users via several mechanisms (Section 3). Almost all products are delivered via source code to at least some of their users. Each of the major DOE computing facilities provides direct support of some users for about 20 ECP ST products. About 10 products are available via vendor software stack and via binary distributions such as Linux distributions.

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

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

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

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

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

ECP ST is developing a software stack to meet the needs of a broad set of Exascale applications. The current software portfolio covers many projects spanning the areas of programming models and runtimes, development tools, mathematical libraries and frameworks, data management, analysis and visualization, and software delivery. The ECP software stack was developed bottom up based on application requirements and the existing software stack at DOE HPC Facilities. The portfolio comprises projects selected in two different ways:

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

2. Three DOE NNSA/ASC funded projects that are part of the Advanced Technology Development and Mitigation (ATDM) program, which is in its sixth year (started in FY14). These projects are focused on longer term research to address the shift in computing technology to extreme, heterogeneous architectures and to advance the capabilities of NNSA/ASC simulation codes.

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

1.1 BACKGROUND

Historically, the software used on supercomputers has come from three sources: computer system vendors, DOE laboratories, and academia. Traditionally, vendors have supplied system software: operating system, compilers, runtime, and system-management software. The basic system software is typically augmented by software developed by the DOE HPC facilities to fill gaps or to improve management of the systems. An observation is that it is common for system software to break or not perform well when there is a jump in the scale of the system.

Mathematical libraries and tools for supercomputers have traditionally been developed at DOE laboratories and universities and ported to the new computer architectures when they are deployed. Vendors also play a role in this space by optimizing the implementations of commonly-used libraries and tools for their architectures, while retaining the interfaces defined by the broader community. This approach enables compile and link time replacement to improve performance on a specific platform by using the vendor versions. Math libraries and tools have been remarkably robust and have supplied some of the most impactful improvements in application performance and productivity. The challenges have been the constant adapting and tuning to rapidly changing architectures.

Programming paradigms and the associated programming environments that include compilers, debuggers, message passing, and associated runtimes have traditionally been developed by vendors, DOE laboratories, and universities. The same can be said for file system and storage software. An observation is that the vendor

WBS 2.3.1	Programming Models and Runtimes	Cross-platform, production-ready programming infrastructure to support development and scaling of mission-critical software at both the node and full-system levels.
WBS 2.3.2	Development Tools	A suite of tools and supporting unified infrastructure aimed at improving developer productivity across the software stack. This scope includes debuggers, profilers, and the supporting compiler infrastructure, with a particular emphasis on LLVM [5] as a delivery and deployment vehicle.
WBS 2.3.3	Mathematical Libraries	Mathematical libraries and frameworks that (i) interoperate with the ECP software stack; (ii) are incorporated into ECP applications; and (iii) provide scalable, resilient numerical algorithms that facilitate efficient simulations on Exascale computers.
WBS 2.3.4	Data and Visualization	Production infrastructure necessary to manage, share, and facilitate analysis and visualization of data in support of mission-critical codes. Data analytics and visualization software that supports scientific discovery and understanding, despite changes in hardware architecture and the size, scale, and complexity of simulation and performance data produced by Exascale platforms.
WBS 2.3.5	Software Ecosystem and Delivery	Development and coordination of Software Development Kits (SDKs), the Extreme-scale Scientific Software Stack (E4S) across all of ECP ST projects. Development of capabilities in Spack [1] in collaboration with NNSA's primary sponsorship. Development of SuperContainers [6] and coordination of container-based workflows across DOE computing facilities.
WBS 2.3.6	NNSA ST	Development and enhancement of open source software capabilities that are primarily developed at Lawrence Livermore, Los Alamos and Sandia National Laboratories. Funds for engaging open science application and software teams in the use and enhancement of these products.

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

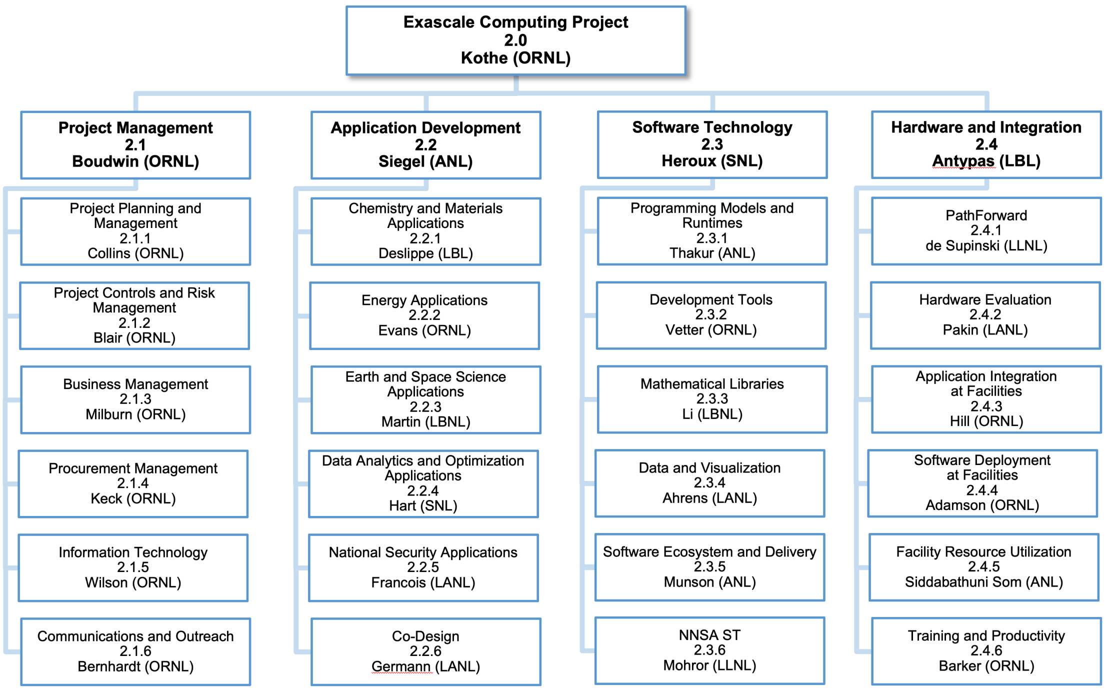


Figure 1: The ECP Work Breakdown Structure through Level 3 (L3) as of December 5, 2019. Under Software Technology, WBS 2.3.6 consolidates ATDM contributions to ECP into a new L3 area.

is ultimately responsible for providing a programming environment and file system with the supercomputer, but there is often a struggle to get the vendors to support software developed by others or to invest in new ideas that have few or no users yet. Another observation is that file-system software plays a key role in overall system resilience, and the difficulty of making the file-system software resilient has grown non-linearly with the scale and complexity of the supercomputers.

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

- **Extreme parallelism:** Experience has shown that software breaks at each shift in scale. Exascale systems are predicted to have a billion-way concurrency almost exclusively from discrete accelerator devices, similar to today's GPUs. An alternate approach using many cores with vector units is also competitive, but still requires the same approximate amount of parallelism. Because clock speeds have essentially stalled, the 1000-fold increase in potential performance going from Petascale to Exascale is entirely from concurrency improvements.

- **Data movement in a deep memory hierarchy:** Data movement has been identified as a key impediment to performance and power consumption. Exascale system designs are increasing the types and layers of memory, which further challenges the software to increase data locality and reuse, while reducing data movement.

- **Discrete memory and execution spaces:** The node architectures of Exascale systems include host CPUs and discrete device accelerators. Programming for these systems requires coordinated transfer of data and work between the host and device. While some of this transfer can be managed implicitly, for the most performance-sensitive phases, the programmer typically must manage host-device coordination explicitly. Much of the software transformation effort will be focused on this issue.

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

- **2021 acceleration:** ECP has a goal of accelerating the development of the U.S. Exascale systems and enabling the first deployment by 2021. This means that the software needs to be ready sooner, and the approach of just waiting until it is ready will not work. A concerted plan that accelerates the development of the highest priority and most impactful software is needed.
- **Productivity:** Traditional supercomputer software requires a great deal of expertise to use. ECP has a goal of making Exascale computing accessible to a wider science community than previous supercomputers have been. This requires the development of software that improves productivity and ease of use.
- **Diversity:** There is a strong push to make software run across diverse Exascale systems. Accelerator devices from Nvidia have been available for many years and specific host-device programming and execution applications have been successfully ported to these platforms. Exascale platforms will continue to have this kind of execution model, but with different programming and runtime software stacks. Writing high-performance, portable code for these platforms will be challenging.
- **Analytics and machine learning:** Future DOE supercomputers will need to solve emerging data science and machine learning problems in addition to the traditional modeling and simulation applications. This will require the development of scalable, parallel analytics and machine learning software for scientific applications, much of which does not exist today.

The next section describes the approach employed by ECP ST to address the Exascale challenges.

1.2 ECP ST PROJECT WBS CHANGES

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

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

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

2.1 ECP SOFTWARE TECHNOLOGY ARCHITECTURE AND DESIGN

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

The portfolio of projects in ECP ST is intended to address the Exascale challenges and requirements described above. We note that ECP is not developing the entire software stack for an Exascale system. For example, we expect vendors to provide the core software that comes with the system (in many cases, by leveraging ECP and other open-source efforts). Examples of vendor-provided software include operating

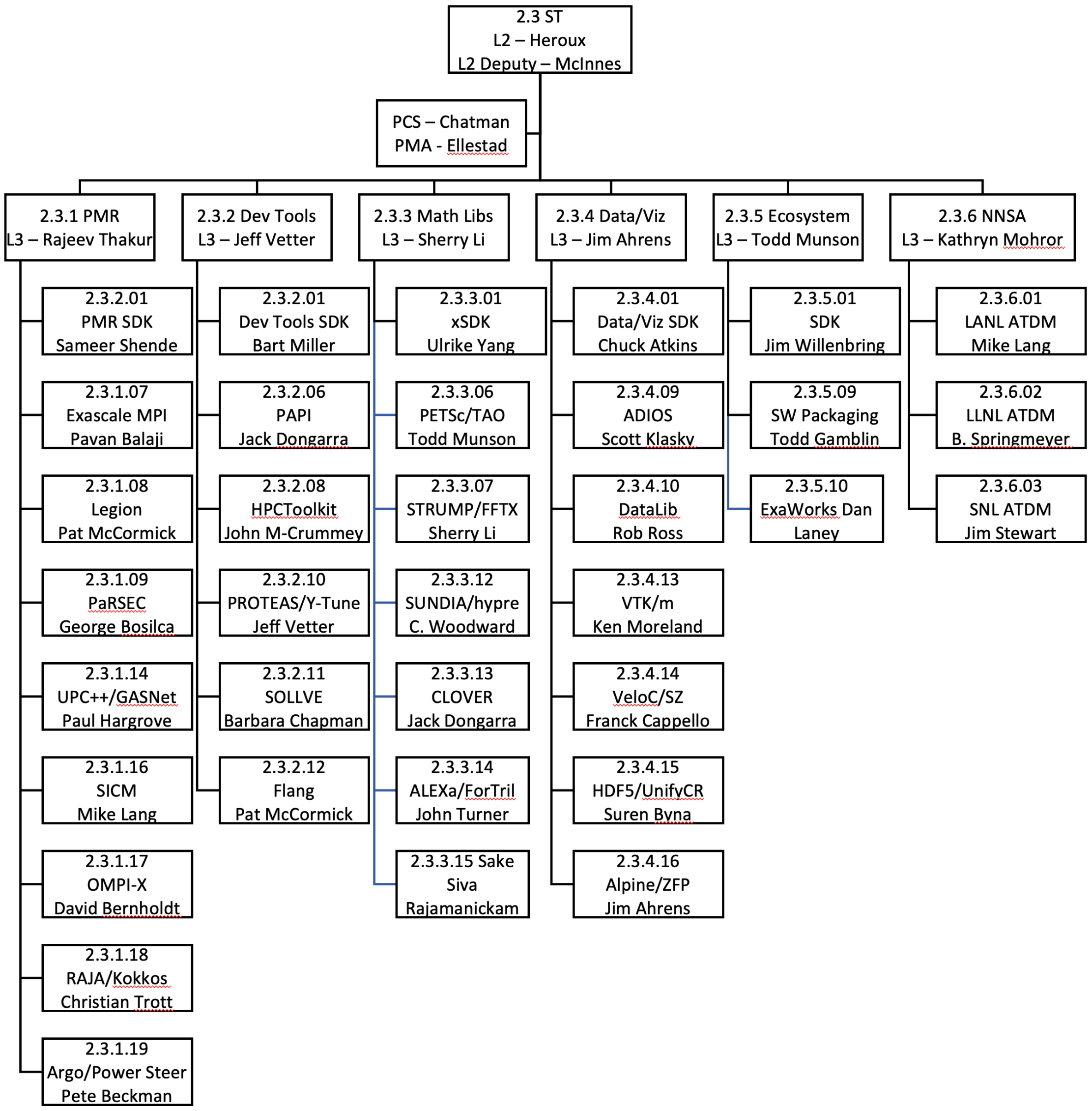

2.3 ST
L2 – Heroux
L2 Deputy – McInnes
PCS – Chatman
PMA - Ellestad
2.3.1 PMR
L3 – Rajeev Thakur
2.3.2 Dev Tools
L3 – Jeff Vetter
2.3.3 Math Libs
L3 – Sherry Li
2.3.4 Data/Viz
L3 – Jim Ahrens
2.3.5 Ecosystem
L3 – Todd Munson
2.3.6 NNSA
L3 – Kathryn Mohror
2.3.2.01 PMR SDK Sameer Shende
2.3.1.07 Exascale MPI Pavan Balaji
2.3.1.08 Legion Pat McCormick
2.3.1.09 PaRSEC George Bosilca
2.3.1.14 UPC++/GASNet Paul Hargrove
2.3.1.16 SICM Mike Lang
2.3.1.17 OMPI-X David Bernholdt
2.3.1.18 RAJA/Kokkos Christian Trott
2.3.1.19 Argo/Power Steer Pete Beckman
2.3.2.01 Dev Tools SDK Bart Miller
2.3.2.06 PAPI Jack Dongarra
2.3.2.08 HPCToolkit John M-Crummey
2.3.2.10 PROTEAS/Y-Tune Jeff Vetter
2.3.2.11 SOLLVE Barbara Chapman
2.3.2.12 Flang Pat McCormick
2.3.3.01 xSDK Ulrike Yang
2.3.3.06 PETSc/TAO Todd Munson
2.3.3.07 STRUMP/FFTX Sherry Li
2.3.3.12 SUNDIA/hypre C. Woodward
2.3.3.13 CLOVER Jack Dongarra
2.3.3.14 ALEXa/ForTril John Turner
2.3.3.15 Sake Siva Rajamanickam
2.3.4.01 Data/Viz SDK Chuck Atkins
2.3.4.09 ADIOS Scott Klasky
2.3.4.10 DataLib Rob Ross
2.3.4.13 VTK/m Ken Moreland
2.3.4.14 VeloC/SZ Franck Cappello
2.3.4.15 HDF5/UnifyCR Suren Byna
2.3.4.16 Alpine/ZFP Jim Ahrens
2.3.5.01 SDK Jim Willenbring
2.3.5.09 SW Packaging Todd Gamblin
2.3.5.10 ExaWorks Dan Laney
2.3.6.01 LANL ATDM Mike Lang
2.3.6.02 LLNL ATDM B. Springmeyer
2.3.6.03 SNL ATDM Jim Stewart

Figure 2: The FY20 ECP ST WBS structure as of November 18, 2020, includes two new L4 subprojects: 2.3.5.10 ExaWorks, a workflow components project, and 2.3.3.15 Sake, a new solver effort that provides funding for Trilinos porting to Frontier and Aurora platforms.

- Phase 1: 66 total L4 subprojects
  - 35 projects funded by the DOE Office of Science that were selected in late 2016 via an RFI and RFP process, considering prioritized requirements of applications and DOE facilities. These projects started work in January–March 2017 depending on when the contracts were awarded.
  - 31 ongoing DOE NNSA funded projects that are part of the Advanced Technology Development and Mitigation (ATDM) program. The ATDM program started in FY14. These projects are focused on longer term research to address the shift in computing technology to extreme, heterogeneous architectures and to advance the capabilities of NNSA simulation codes.
- Phase 2: 55 total L4 subprojects
  - 41 ASCR-funded projects. Added 2 SW Ecosystem & Delivery projects and 4 SDK projects.
  - 15 ATDM projects: Combined the previous 31 ATDM projects into one project per technical area per lab. ATDM projects are generally more vertically integrated and would not perfectly map to any proposed ECP ST technical structure. Minimizing the number of ATDM projects within the ECP WBS structure reduces complexity of ATDM to ECP coordination and gives ATDM flexibility in revising its portfolio without disruption to the ECP-ATDM mapping.
- Phase 3a: 33 total L4 subprojects. Fewer, larger and more uniform-sized projects
  - Starting with FY2020, ECP ST has further consolidated L4 projects to foster additional synergies and amortize project overheads as ECP heads into Critical Decision Phase 2 [7], where more rigor in planning and execution are needed.
  - 5 L3s to 6: New NNSA ST L3
  - 40 ST SC-funded L4 subprojects to 30.
    - Programming Models & Runtimes– 13 to 9, Development Tools- 6 to 6, Mathematical Libraries- 7 to 6, Data & Visualization- 10 to 7, SW Ecosystem & Delivery- 4 to 3.
    - Includes 2 new L4 subprojects in SW Ecosystem & Delivery.
  - 15 ST NNSA-funded projects transferred to new NNSA ST L3. Consolidated from 15 to 3 L4 subprojects.
  - No more small subprojects.
  - Figure 2 show the overall structure.
- Phase 3b: 35 total L4 subprojects. Add two new L4 subprojects.
  - New L4 subproject called ExaWorks. Focuses on providing an underlying component architecture for workflow management systems, led by a team of workflow experts who would leverage the new substrate in their own workflow products.
  - New L4 subproject called Sake. This project was created in response to a need for Trilinos funding to port to Aurora and Frontier. At the same time, Trilinos-related activities in the CLOVER project, specifically Kokkos Kernels, were merged with the new Trilinos funding to create a more holistic project, independent of CLOVER.
  - Figure 2 show the overall structure.

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

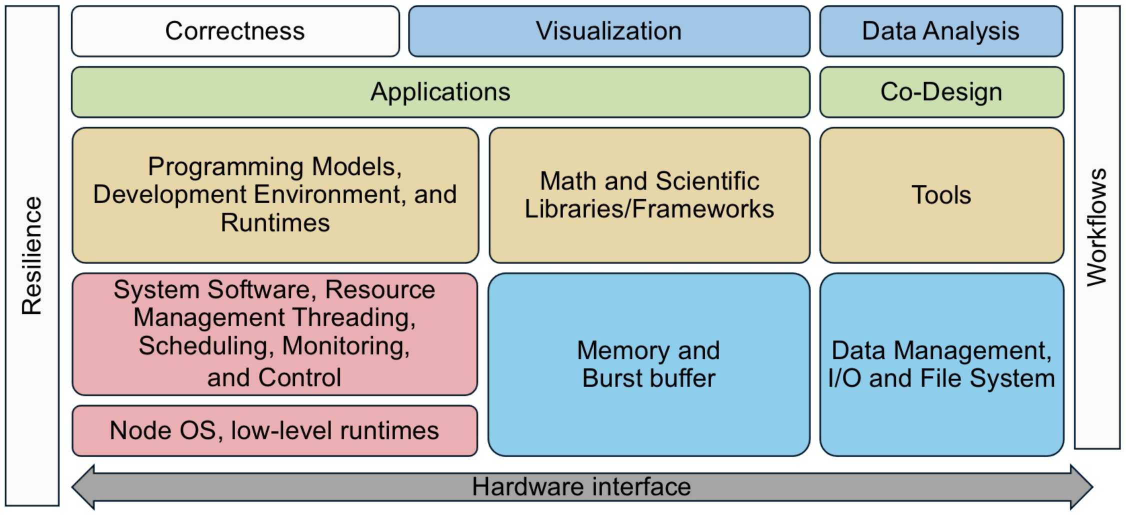

Correctness
Visualization
Data Analysis
Applications
Co-Design
Resilience
Programming Models, Development Environment, and Runtimes
Math and Scientific Libraries/Frameworks
Tools
Workflows
System Software, Resource Management Threading, Scheduling, Monitoring, and Control
Memory and Burst buffer
Data Management, I/O and File System
Node OS, low-level runtimes
Hardware interface

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

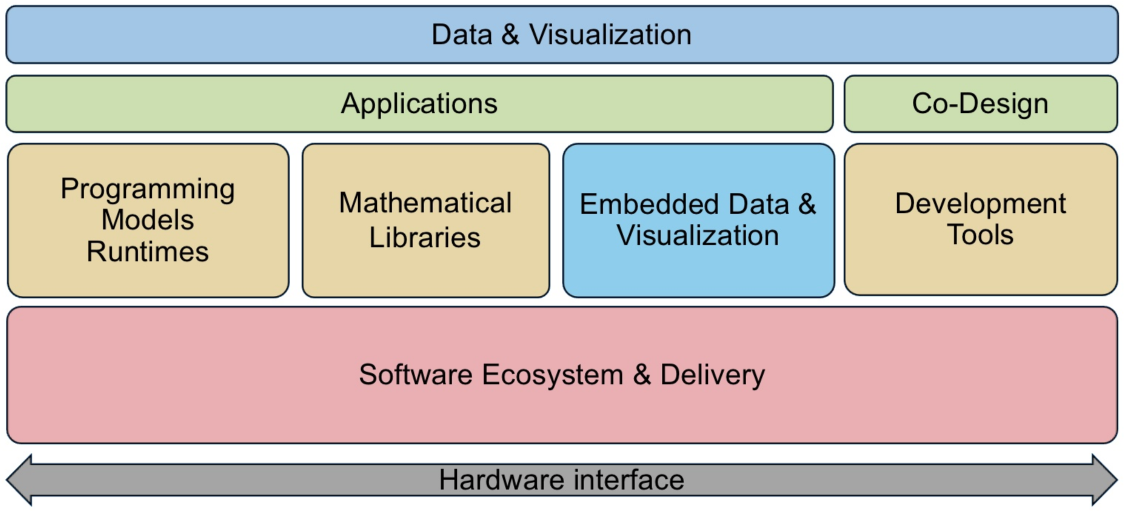


Figure 5: ECP ST after November 2017 reorganization. This diagram more accurately reflects the priorities and efforts of ECP ST given the new ECP project scope and the demands that we foresee.

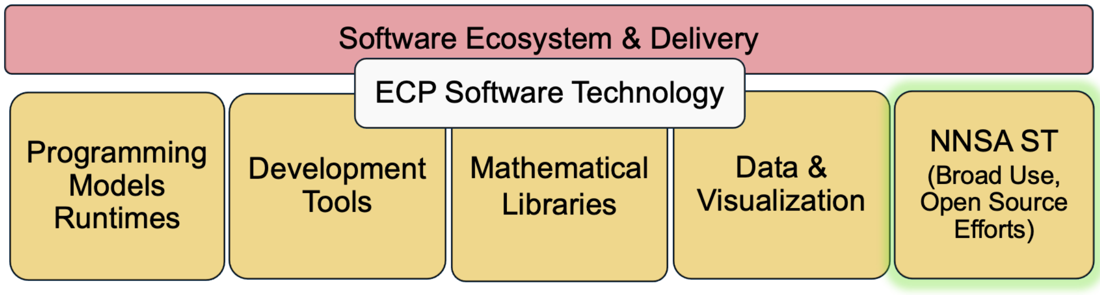
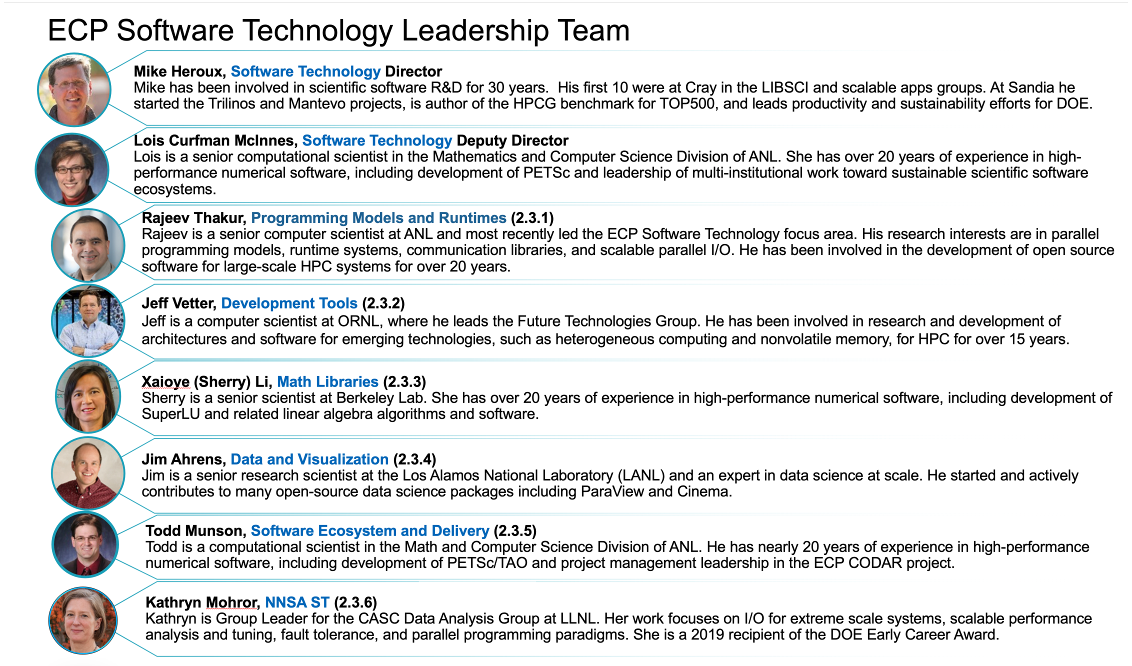

Software Ecosystem & Delivery
ECP Software Technology
Programming Models Runtimes
Development Tools
Mathematical Libraries
Data & Visualization
NNSA ST (Broad Use, Open Source Efforts)

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



ECP Software Technology Leadership Team

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

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

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

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

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

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

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

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

Figure 7: ECP ST Leadership Team as of November 2020. Jonathan Carter, previous Deputy Director of ECP ST, became Associate Lab Director of the Computing Sciences Area at Lawrence Berkeley National Laboratory. His departure led to naming of Lois Curfman McInnes, previously the L3 Lead of Math Libraries, as Deputy Director of ECP ST, and Sherry Li as the new Math Libraries L3 Lead. Rob Neely was also promoted at LLNL, leading to Kathryn Mohror becoming the L3 Lead of NNSA ST.

system, file system, compilers for C, C++, Fortran, etc. (increasingly derived from the LLVM community ecosystem to which ECP contributes), basic math libraries, system monitoring tools, scheduler, debuggers, vendor's performance tools, MPI (based on ECP-funded projects), OpenMP (with features from ECP-funded project), and data-centric stack components. ECP develops other, mostly higher-level software that is needed by applications and is not vendor specific. ECP-funded software activities are concerned with extreme scalability, exposing additional parallelism, unique requirements of Exascale hardware, and performance-critical components. Other software that aids in developer productivity is needed and may come from third-party open-source efforts.

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

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

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

2.1.1 The Extreme-scale Scientific Software Stack (E4S)

In October 2020, ECP ST released version 1.2 of the Extreme-scale Scientific Software Stack, E4S (`http://e4s.io`). E4S contains a collection of the software products to which ECP ST contributes. E4S is the primary conduit for providing easy access to ECP ST capabilities for ECP and the broader community. E4S is also the ECP ST vehicle for regression and integration testing across DOE pre-Exascale and Exascale systems.

E4S has the following key features:

- **The E4S suite is a large and growing effort to build and test a comprehensive scientific software ecosystem:** In November 2018, E4S V0.1 contained 25 ECP products. Two years later, E4S V1.2, the fifth E4S release, contained 67 ECP ST products and numerous additional products needed for a complete software environment. Eventually E4S will contain all open source products to which ECP contributes, and all related products needed for a holistic environment.

- **E4S is not an ECP-specific software suite:** The products in E4S represent a holistic collection of capabilities that contain the ever-growing SDK collections sponsored by ECP and all additional underlying software required to use ECP ST capabilities. Furthermore, we expect the E4S effort to live beyond the timespan of ECP, becoming a critical element of the scientific software ecosystem.

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

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

- **E4S is available via containers:** In addition to a build-from-source capability using Spack, E4S maintains several container environments (Docker, Singularity, Shifter, CharlieCloud) that provides the lowest barrier to use. Container distributions dramatically reduce installation costs and provide a ready-made environment for tutorials that leverage E4S capabilities. For example, E4S containers now support custom images for ECP applications such as WDMapp and Pantheon.

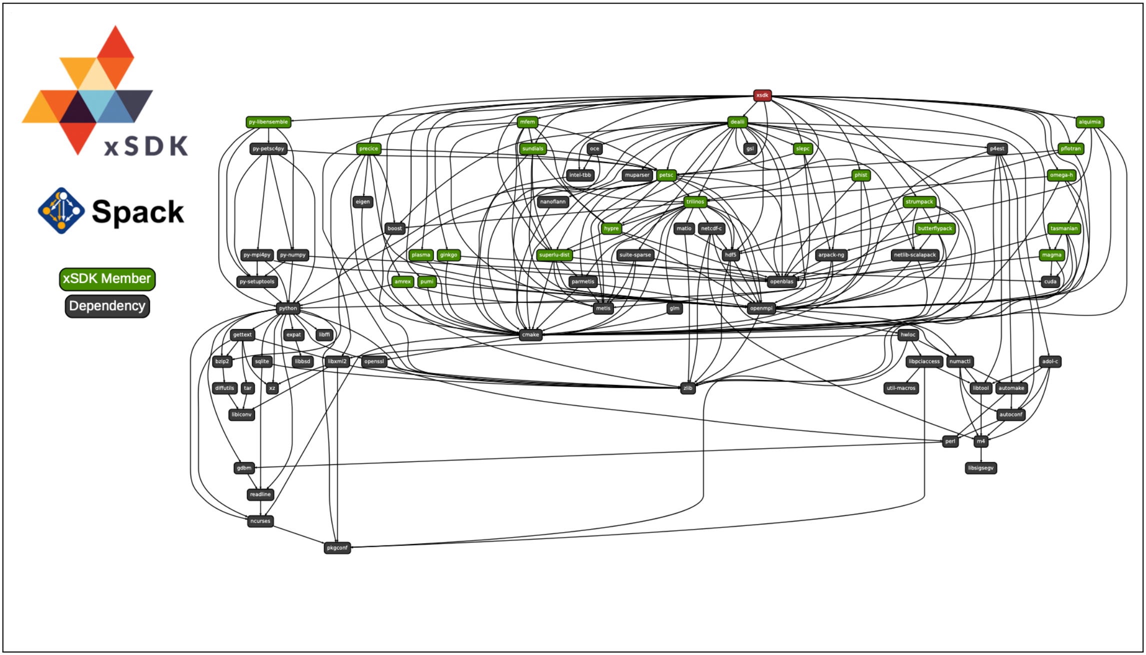


Figure 8: Using Spack [2], E4S builds a comprehensive software stack. As ECP ST efforts proceed, we will use E4S for continuous integration testing, providing developers with rapid feedback on regression errors and providing user facilities with a stable software base as we prepare for Exascale platforms. This diagram shows how E4S builds ECP products via an SDK target (the math libraries SDK called xSDK in this example). The SDK target then builds all product that are part of the SDK (see Figure 13 for SDK groupings), first defining and building external software products. Green-labeled products are part of the SDK. The blue label indicates expected system tools, in this case a particular version of Python. Black-labeled products are expected to be previously installed into the environment (a common requirement and easily satisfied). Using this approach, a user who is interested in only SUNDIALS (a particular math library) can be assured that the SUNDIALS build will be possible since it is a portion of what E4S builds and tests.

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

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

- **E4S distribution:** E4S products are available at `http://e4s.io`.
- **E4S developer community resources:** Developers interested in participating in E4S can visit the E4S-Project GitHub community at `https://github.com/E4S-Project`.

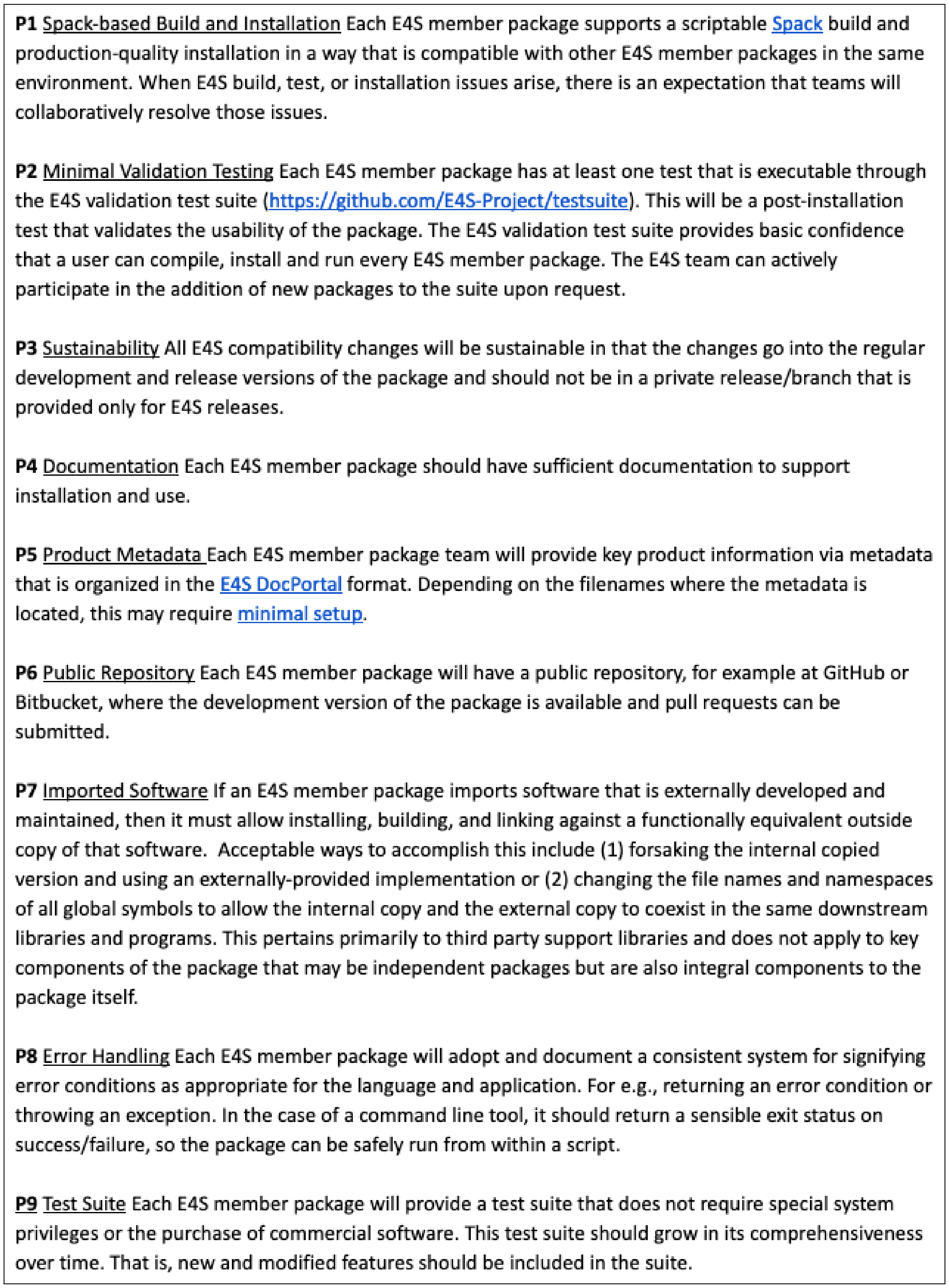
The first set of E4S Community Policies [8] was adopted in October 2020 (see Figure 9). These Policies are membership criteria for a product to become an E4S member package. The purpose of the Community Policies is to establish baseline software quality and practice expectations to help address sustainability and interoperability challenges for the Software Technologies software ecosystem. While a package does not have to demonstrate compatibility with the policies as a condition of inclusion in E4S releases, compatibility is necessary for member package designation.

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

2.1.2 Software Development Kits

One opportunity for a large software ecosystem project such as ECP ST is to foster increased collaboration, integration and interoperability among its funded efforts. Part of ECP ST design is the creation of software development kits (SDKs). SDKs are collections of related software products (called packages) where coordination across package teams will improve usability and practices and foster community growth among teams that develop similar and complementary capabilities. SDKs have the following attributes:

ECP ST SDKs As part of the delivery of ECP ST capabilities, we will establish and grow a collection of SDKs. The new layer of aggregation that SDKs represent are important for improving all aspects of product development and delivery. The communities that will emerge from SDK efforts will lead to better collaboration and higher quality products. Established community policies will provide a means to grow



P1 Spack-based Build and Installation Each E4S member package supports a scriptable Spack build and production-quality installation in a way that is compatible with other E4S member packages in the same environment. When E4S build, test, or installation issues arise, there is an expectation that teams will collaboratively resolve those issues.

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

P3 Sustainability All E4S compatibility changes will be sustainable in that the changes go into the regular development and release versions of the package and should not be in a private release/branch that is provided only for E4S releases.

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

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

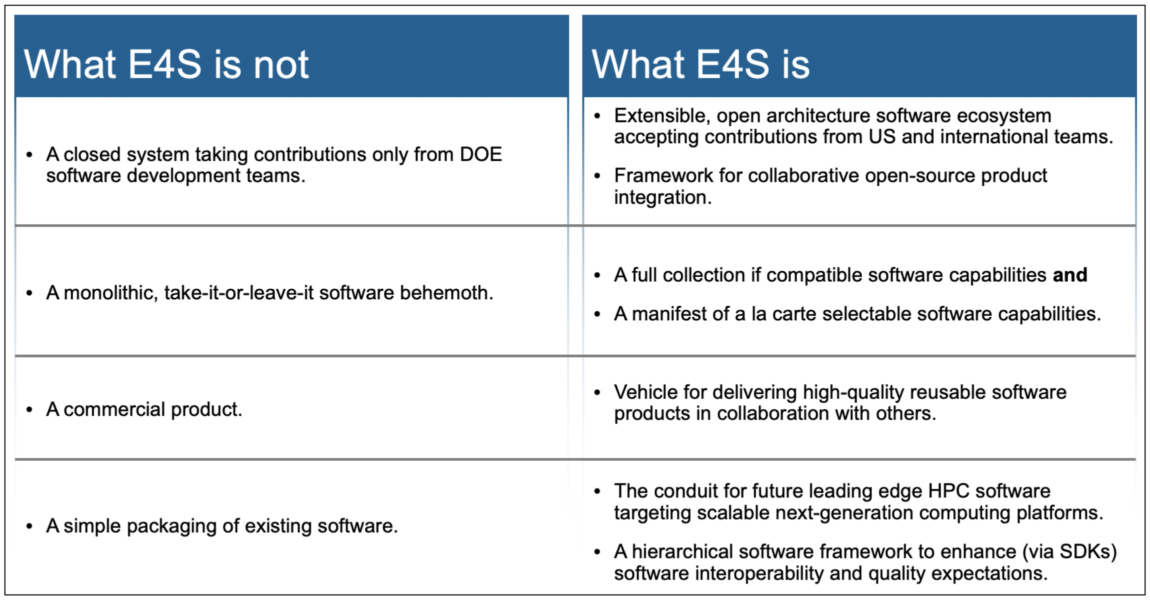
P6 Public Repository Each E4S member package will have a public repository, for example at GitHub or Bitbucket, where the development version of the package is available and pull requests can be submitted.

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

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

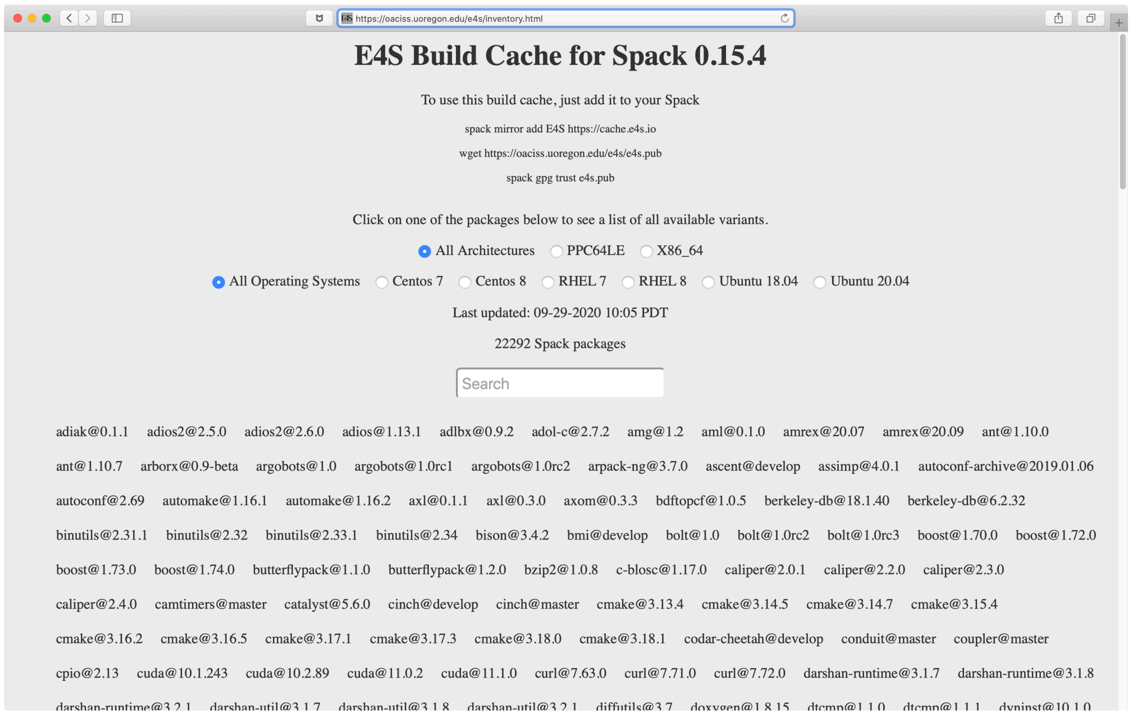
P9 Test Suite Each E4S member package will provide a test suite that does not require special system privileges or the purchase of commercial software. This test suite should grow in its comprehensiveness over time. That is, new and modified features should be included in the suite.

Figure 9: Version 1 of the E4S Community Policies. These policies will serve as membership criteria for E4S member packages. The E4S Community Policy effort has heavily leveraged the existing xSDK Community Policies [3].



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

Figure 10: The Extreme-scale Scientific Software Stack (E4S) provides a complete Linux-based software stack that is suitable for many scientific workloads, tutorial and development environments. At the same time, it is an open software architecture that can expand to include any additional and compatible Spack-enabled software capabilities. Since Spack packages are available for many products and easily created for others, E4S is practically expandable to include almost any robust Linux-based product. Furthermore, E4S capabilities are available as subtrees of the full build: E4S is not monolithic.



E4S Build Cache for Spack 0.15.4

To use this build cache, just add it to your Spack

```
spack mirror add E4S https://cache.e4s.io
wget https://oaciss.uoregon.edu/e4s/e4s.pub
spack gpg trust e4s.pub
```

Click on one of the packages below to see a list of all available variants.

[✓ All Architectures] PPC64LE X86_64

[✓ All Operating Systems] Centos 7 Centos 8 RHEL 7 RHEL 8 Ubuntu 18.04 Ubuntu 20.04

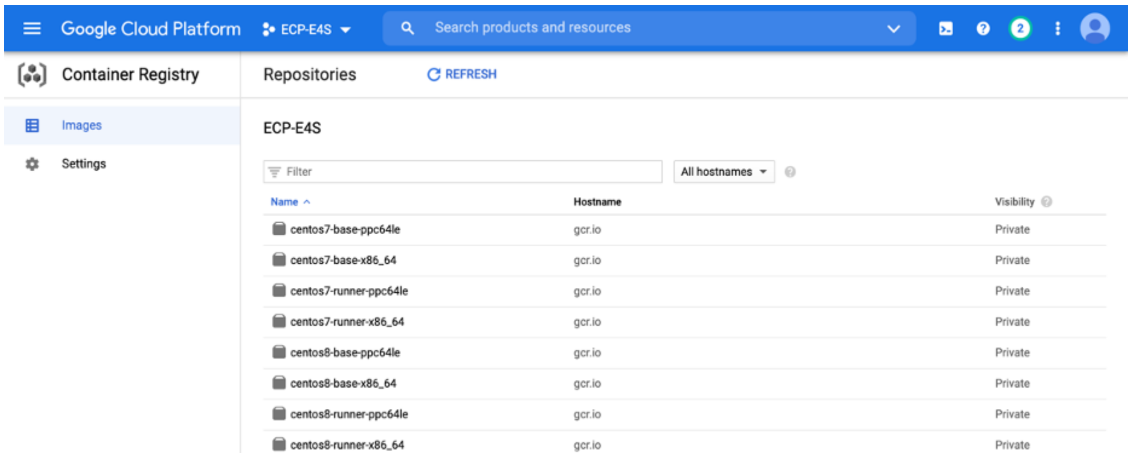
Last updated: 09-29-2020 10:05 PDT

22292 Spack packages

Search

```
adiak@0.1.1  adios2@2.5.0  adios2@2.6.0  adios@1.13.1  adlbx@0.9.2  adol-c@2.7.2  amg@1.2  aml@0.1.0  amrex@20.07  amrex@20.09  ant@1.10.0
ant@1.10.7  arborx@0.9-beta  argobots@1.0  argobots@1.0rc1  argobots@1.0rc2  arpack-ng@3.7.0  ascent@develop  assimp@4.0.1  autoconf-archive@2019.01.06
autoconf@2.69  automake@1.16.1  automake@1.16.2  axl@0.1.1  axl@0.3.0  axom@0.3.3  bdftopcf@1.0.5  berkeley-db@18.1.40  berkeley-db@6.2.32
binutils@2.31.1  binutils@2.32  binutils@2.33.1  binutils@2.34  bison@3.4.2  bmi@develop  bolt@1.0  bolt@1.0rc2  bolt@1.0rc3  boost@1.70.0  boost@1.72.0
boost@1.73.0  boost@1.74.0  butterflypack@1.1.0  butterflypack@1.2.0  bzip2@1.0.8  c-blosc@1.17.0  caliper@2.0.1  caliper@2.2.0  caliper@2.3.0
caliper@2.4.0  camtimers@master  catalyst@5.6.0  cinch@develop  cinch@master  cmake@3.13.4  cmake@3.14.5  cmake@3.14.7  cmake@3.15.4
cmake@3.16.2  cmake@3.16.5  cmake@3.17.1  cmake@3.17.3  cmake@3.18.0  cmake@3.18.1  codar-cheetah@develop  conduit@master  coupler@master
cpio@2.13  cuda@10.1.243  cuda@10.2.89  cuda@11.0.2  cuda@11.1.0  curl@7.63.0  curl@7.71.0  curl@7.72.0  darshan-runtime@3.1.7  darshan-runtime@3.1.8
darshan-runtime@3.2.1  darshan-util@3.1.7  darshan-util@3.1.8  darshan-util@3.2.1  diffutils@3.7  doxygen@1.8.15  dtcmp@1.1.0  dtcmp@1.1.1  dyninst@10.1.0
```

Figure 11: Using Spack build cache features, E4S builds can be accelerated by use of cached binaries for any build signature that Spack has already seen. Between September 2019 and September 2020, more than 21,000 binaries were added to the cache.



Name	Hostname	Visibility
centos7-base-ppc64le	gcr.io	Private
centos7-base-x86_64	gcr.io	Private
centos7-runner-ppc64le	gcr.io	Private
centos7-runner-x86_64	gcr.io	Private
centos8-base-ppc64le	gcr.io	Private
centos8-base-x86_64	gcr.io	Private
centos8-runner-ppc64le	gcr.io	Private
centos8-runner-x86_64	gcr.io	Private

Figure 12: E4S now supports Google Cloud Platform in addition to Amazon AWS.

SDKs beyond ECP to include any relevant external effort. The meta-build systems (based on Spack) will play an important role in managing the complexity of building the ECP ST software stack, by providing a new layer where versioning, consistency and build options management can be addressed at a mid-scope, below the global build of ECP ST products. Each ECP ST L3 (five of them) has funds for an SDK project from which we have identified a total of six SDKs and an at-large collection of remaining products that will be delivered outside of the SDK grouping. Section 4.5.7 provides an update on the progress in defining SDK groupings. For visibility, we provide the same diagram in Figure 13.

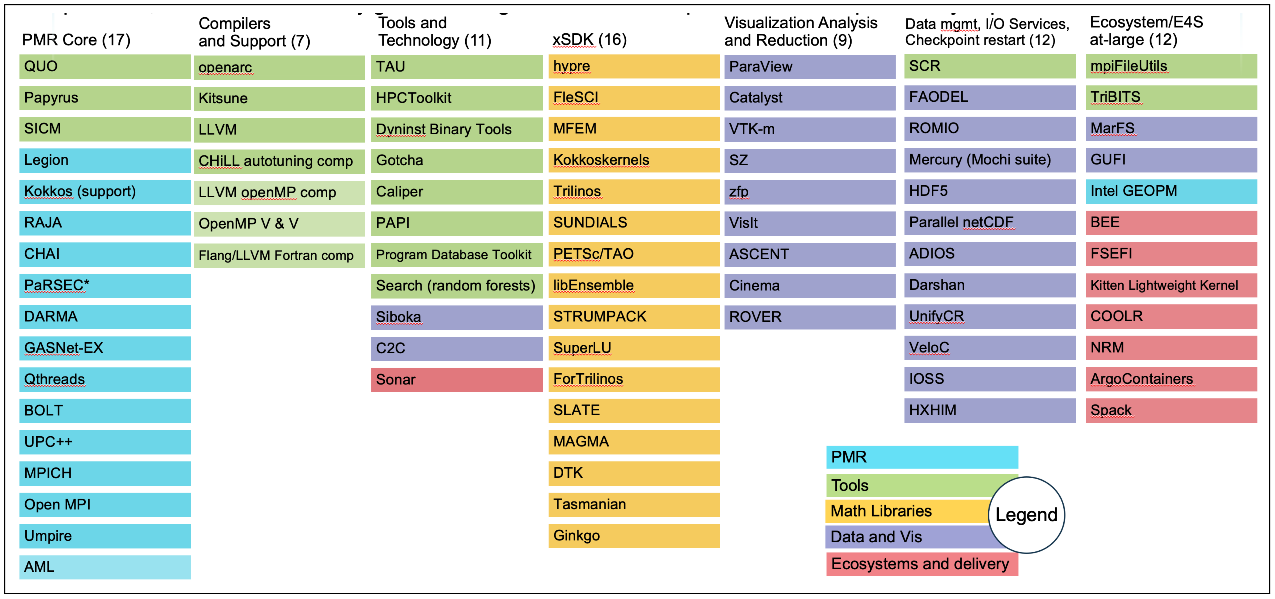
2.1.3 ECP ST Product Dictionary

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

Figure 14 describes the policy for ECP ST teams to add and manage their contributions to the Product Dictionary. Figure 15 shows a snapshot of the beginning and end of the current ECP ST Product Dictionary, which is maintained on the ECP Confluence wiki.

2.1.4 ECP Product Dependency Management

Given the ECP ST Product Dictionary, and a similar dictionary for ECP AD and Co-Design products, ECP as a project has created a dependency database that enabled creation and characterization of product-to-product dependencies. ECP manages these dependencies in a Jira database using a custom Jira issue type, Dependency. The dependency database provides an important tool for understanding and managing ECP activities. The dependency information is valuable both within and outside the project. Figure



PMR Core (17)	Compilers and Support (7)	Tools and Technology (11)	xSDK (16)	Visualization Analysis and Reduction (9)	Data mgmt, I/O Services, Checkpoint restart (12)	Ecosystem/E4S at-large (12)
QUO	openarc	TAU	hypre	ParaView	SCR	mpiFileUtils
Papyrus	Kitsune	HPCToolkit	FleSCI	Catalyst	FAODEL	TriBITS
SICM	LLVM	Dyninst Binary Tools	MFEM	VTK-m	ROMIO	MarFS
Legion	CHiLL autotuning comp	Gotcha	Kokkoskernels	SZ	Mercury (Mochi suite)	GUFI
Kokkos (support)	LLVM openMP comp	Caliper	Trilinos	zfp	HDF5	Intel GEOPM
RAJA	OpenMP V & V	PAPI	SUNDIALS	VisIt	Parallel netCDF	BEE
CHAI	Flang/LLVM Fortran comp	Program Database Toolkit	PETSc/TAO	ASCENT	ADIOS	FSEFI
PaRSEC*		Search (random forests)	libEnsemble	Cinema	Darshan	Kitten Lightweight Kernel
DARMA		Siboka	STRUMPACK	ROVER	UnifyCR	COOLR
GASNet-EX		C2C	SuperLU		VeloC	NRM
Qthreads		Sonar	ForTrilinos		IOSS	ArgoContainers
BOLT			SLATE		HXHIM	Spack
UPC++			MAGMA			
MPICH			DTK			
Open MPI			Tasmanian			
Umpire			Ginkgo			
AML						

Legend
PMR
Tools
Math Libraries
Data and Vis
Ecosystems and delivery

Figure 13: The above graphic shows the breakdown of ECP ST products into 6 SDKs (the first six columns). The rightmost column lists products that are not part of an SDK, but are part of Ecosystem group that will also be delivered as part of E4S. The colors denoted in the key map all of the ST products to the ST technical area they are part of. For example, the xSDK consists of products that are in the Math Libraries Technical area, plus TuckerMPI which is in the Ecosystem and Delivery technical area. Section 4.5.7 provides an update on the progress in defining SDK groupings.

2.2 ECP ST PLANNING AND TRACKING

While ECP is an official 413.3b federal construction project using an earned value management (EVM) structure, we are permitted to tailor the planning process in order to obtain the flexibility needed for a software project whose requirements are emerging as the project proceeds. In this section, we describe how ECP ST plans it activities using the Jira project management tool. We first discuss P6 Activities (similar to milestones) and then discuss the key performance parameter (KPP-3) associated with ECP ST.

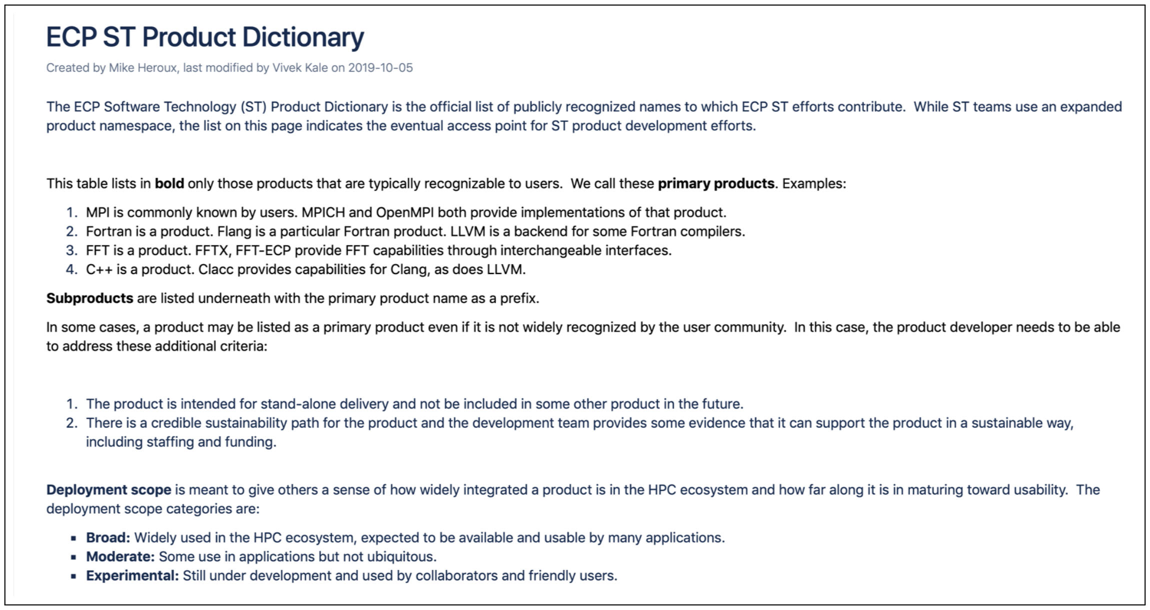
2.2.1 ECP ST P6 Activity Issues

ECP ST uses a custom Jira issue type called P6 Activity. Each L4 subproject creates a series of P6 Activity issues extending to the end of ECP (Q3FY23). Except for the current fiscal year, a single P6 Activity issue describes expected deliverables as a planning package. Six months prior to the start of a fiscal year, the planning package for the coming year is replaced with 4–6 issues spanning the year with baseline start and end dates, an estimate of the percent annual budget and a high-level description. Eight weeks prior to the start of an activity, full details about staffing, completion criteria and more are added to the issue. Figure 18 show the steps in diagram form.

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

2.2.2 Key Performance Parameter (KPP) 3

ECP has four Key Performance Parameters (KPPs). Figure 19 shows the KPP definitions. KPP-3 is focused on a productive and sustainable software ecosystem. ECP ST is the primary owner of this KPP (along with



ECP ST Product Dictionary

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

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

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

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

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

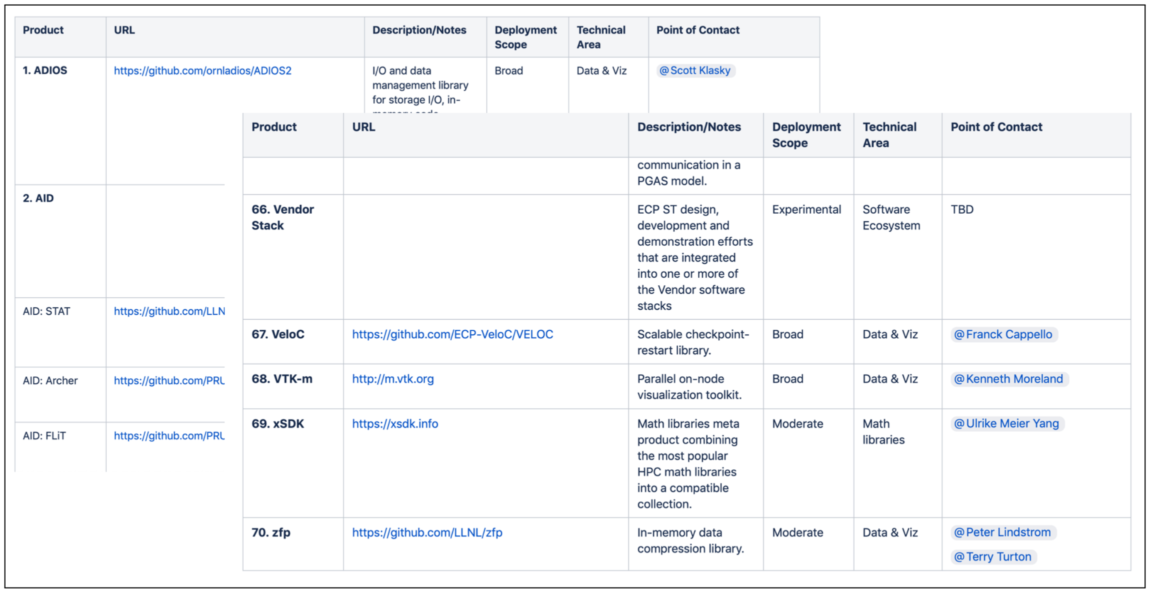
In some cases, a product may be listed as a primary product even if it is not widely recognized by the user community. In this case, the product developer needs to be able to address these additional criteria:

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

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

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

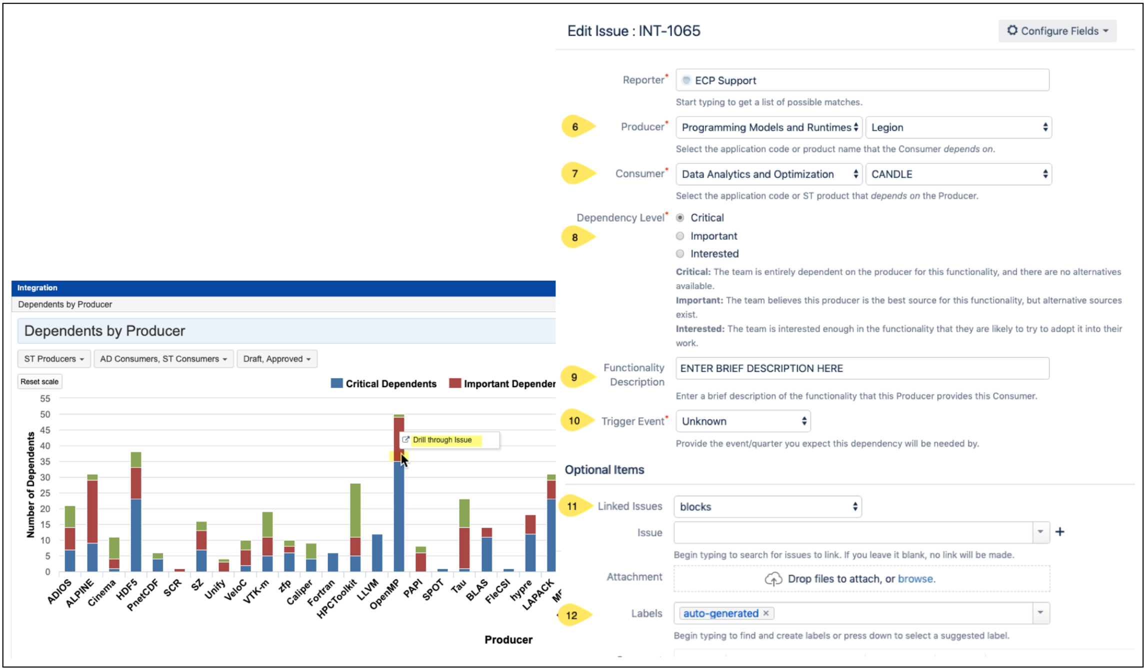
Figure 14: This figure shows a screenshot from the top of the ECP Confluence wiki page containing the ECP ST Product Dictionary. The Product Dictionary structure contains primary and secondary products. Client (consumer) dependencies are stated against the primary product names only, enabling unambiguous mapping of AD-on-ST and ST-on-ST dependencies.



Product	URL	Description/Notes	Deployment Scope	Technical Area	Point of Contact
1. ADIOS	https://github.com/ornladios/ADIOS2	I/O and data management library for storage I/O, in-	Broad	Data & Viz	@Scott Klasky
2. AID					
AID: STAT	https://github.com/LLN				
AID: Archer	https://github.com/PRL				
AID: FLiT	https://github.com/PRL				

Product	URL	Description/Notes	Deployment Scope	Technical Area	Point of Contact
		communication in a PGAS model.			
66. Vendor Stack		ECP ST design, development and demonstration efforts that are integrated into one or more of the Vendor software stacks	Experimental	Software Ecosystem	TBD
67. VeloC	https://github.com/ECP-VeloC/VELOC	Scalable checkpoint-restart library.	Broad	Data & Viz	@Franck Cappello
68. VTK-m	http://m.vtk.org	Parallel on-node visualization toolkit.	Broad	Data & Viz	@Kenneth Moreland
69. xSDK	https://xsdk.info	Math libraries meta product combining the most popular HPC math libraries into a compatible collection.	Moderate	Math libraries	@Ulrike Meier Yang
70. zfp	https://github.com/LLNL/zfp	In-memory data compression library.	Moderate	Data & Viz	@Peter Lindstrom @Terry Turton

Figure 15: These screen shots are from the ECP Confluence Product Dictionary Table. The table is actively managed to include primary and secondary products to which ECP ST team contribute and upon which ECP ST clients depend. Presently the Product Dictionary contains 70 primary products. Secondary products are listed under the primary product with the primary product as a prefix. For example, AID is the second listed primary product in this figure. STAT, Archer and FLIT are component subproducts. MPI (not shown) is another primary product. MPICH and OpenMPI are two robust MPI implementations and are listed as MPI subproducts.



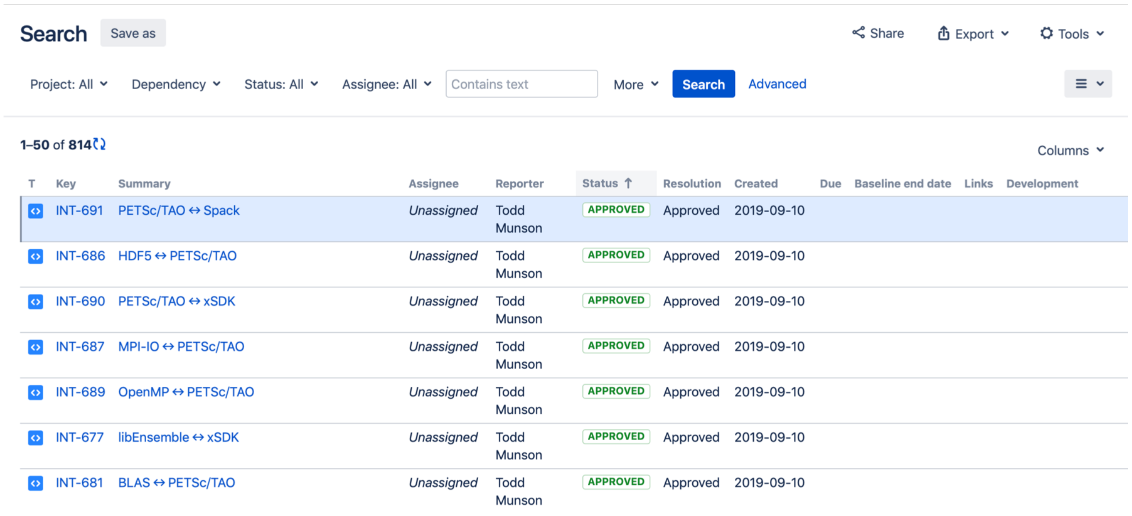
Edit Issue : INT-1065

Reporter*	ECP Support
	Start typing to get a list of possible matches.
Producer*	Programming Models and Runtimes; Legion
	Select the application code or product name that the Consumer *depends on*.
Consumer*	Data Analytics and Optimization; CANDLE
	Select the application code or ST product that *depends on* the Producer.
Dependency Level*	● Critical ○ Important ○ Interested
	Critical: The team is entirely dependent on the producer for this functionality, and there are no alternatives available. **Important:** The team believes this producer is the best source for this functionality, but alternative sources exist. **Interested:** The team is interested enough in the functionality that they are likely to try to adopt it into their work.
Functionality Description	ENTER BRIEF DESCRIPTION HERE
	Enter a brief description of the functionality that this Producer provides this Consumer.
Trigger Event*	Unknown
	Provide the event/quarter you expect this dependency will be needed by.

Optional Items

Linked Issues	blocks
Issue	
	Begin typing to search for issues to link. If you leave it blank, no link will be made.
Attachment	Drop files to attach, or browse.
Labels	auto-generated ×
	Begin typing to find and create labels or press down to select a suggested label.

Figure 16: Using Jira, ECP manages its AD, ST, HI, vendor and facilities dependencies. This figure shows a dashboard snapshot along with an edit panel that support creation and management of a consumer-on-producer dependency.



1–50 of 814

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

Figure 17: This query result from the ECP Jira Dependency database lists all consumers of capabilities from the PETSc/TAO product. By selecting the details of one of the dependency issues, one can further see how critical the dependency is and see any custom information peculiar to the particular dependency.

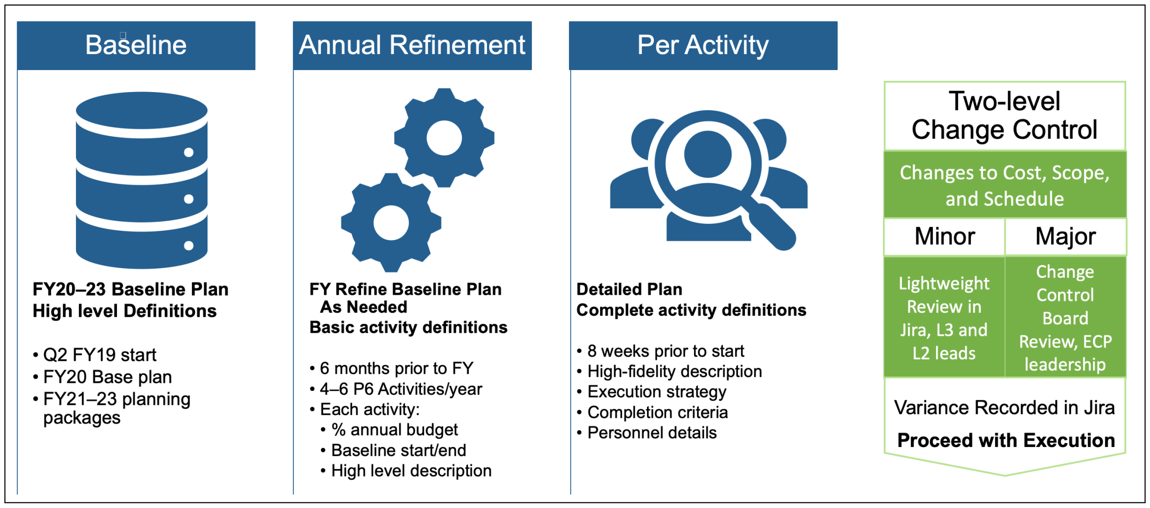


Figure 18: ECP ST uses a custom Jira issue type called P6 Activity. Each L4 subproject creates a series of these issues extending to the end of ECP. Except for the current fiscal year, a single P6 Activity issue describes expected deliverables as a planning package. Six months prior to the start of a fiscal year, the planning package is replaced with 4–6 issues spanning the coming year. Eight weeks prior to the start of an activity, full details about staffing, completion criteria and more are added to the issue.

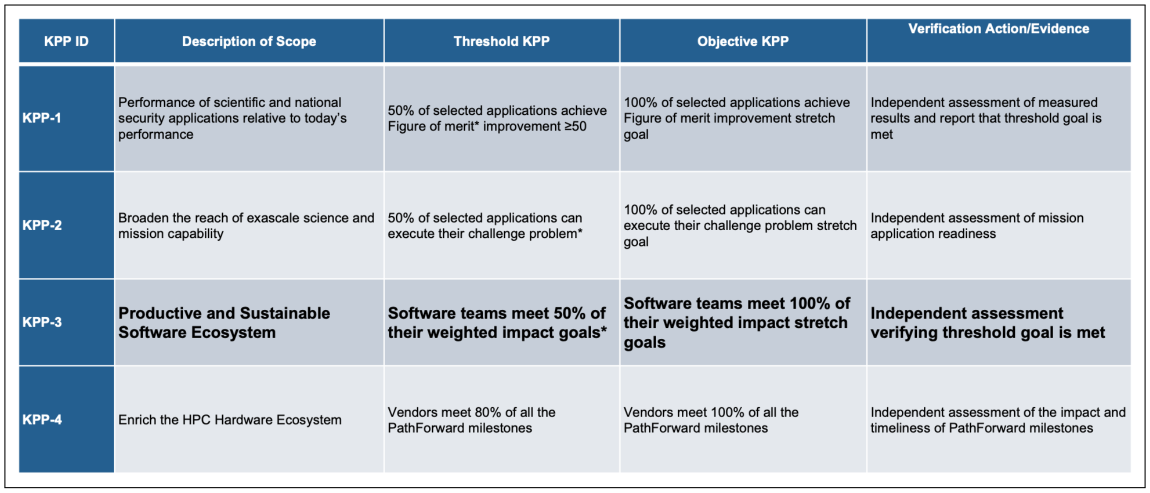
co-design projects in ECP AD). The focus of KPP-3 is defining and tracking capability integrations of ST products into client environment, as described in this section.

First, we define terms:

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

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

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



KPP ID	Description of Scope	Threshold KPP	Objective KPP	Verification Action/Evidence
KPP-1	Performance of scientific and national security applications relative to today's performance	50% of selected applications achieve Figure of merit* improvement ≥50	100% of selected applications achieve Figure of merit improvement stretch goal	Independent assessment of measured results and report that threshold goal is met
KPP-2	Broaden the reach of exascale science and mission capability	50% of selected applications can execute their challenge problem*	100% of selected applications can execute their challenge problem stretch goal	Independent assessment of mission application readiness
KPP-3	**Productive and Sustainable Software Ecosystem**	**Software teams meet 50% of their weighted impact goals***	**Software teams meet 100% of their weighted impact stretch goals**	**Independent assessment verifying threshold goal is met**
KPP-4	Enrich the HPC Hardware Ecosystem	Vendors meet 80% of all the PathForward milestones	Vendors meet 100% of all the PathForward milestones	Independent assessment of the impact and timeliness of PathForward milestones

Figure 19: ECP has four key performance parameters (KPPs). ECP ST is the primary owner (with ECP AD co-design subprojects) of KPP-3.

- A general description of the nature of the integration, addressing what it means to be successfully integrated.

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

Table 3: Integration Goal Scoring: A point is accrued when a client integrates and sustainably uses a product's capabilities. Scores are assessed annually.

Demonstration and recording of progress toward integration goal All artifacts and evidence of progress will be captured in the Jira KPP-3 issue associated with a product integration goal as progress is made. All integration scores are tentative until the capability is available and demonstrated in exascale environments. Table 4 summarizes the defined values.

Assessment process While progress is recorded as it is achieved, progress assessment is done annually, including input from external subject matter experts (SMEs). ECP leadership and SMEs will review integration score evidence, confirming or adjusting integration scores. Note: Assessment can result in a reduced integration score from a previous year if a client has stopped using a capability that was previously used.

Transition from tentative to confirmed integration score Each integration score is tentative until the capability is available and demonstrated to be effective in the exascale environments. Demonstration can be achieved by a variety of means such that ECP Leadership and SMEs are reasonably certain the capability positively impacts the client in exascale environments. At this point the integration score becomes confirmed. Typically, the transition from tentative to confirmed would be a low-cost independent demonstration, or accomplished within the client's environment as the client is conducting its own assessments. Note: The planned exascale system (El Capitan) that can support National Security applications will not be available

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

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

until the end of FY23. Integration of ST products into National Security Applications will be considered for transition from tentative to confirmed when either a) evidence of integration is provided during FY20-22 ASC L1 and L2 milestones related to ECP/ATDM National Security application readiness for exascale platforms, and/or b) integration is demonstrated on the El Capitan early access systems, and exercises capabilities similar to those anticipated to be important to effectively using El Capitan. For KPP-3 capability integrations targeted at El Capitan, we will use the best available confirmation process in FY23. KPP-3 weighted scoring

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

Table 5: Each integration score will have an associated weight depending on the potential impact if integration targets are not met.

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

2.2.3 ECP ST Software Delivery

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

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

- ST **delivers** software. ECP ST products are delivered directly to application teams, to vendors and to facilities. ECP ST designs and implements products to run on DOE computing facilities platforms and make products available as source code via GitHub, GitLab or some other accessible repository.

- HI facilitates efforts to **deploy** ST (and other) software on Facilities platforms by installing it where users expect to find it. This could be in /usr/local/bin or similar directory, or available via "module load".

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

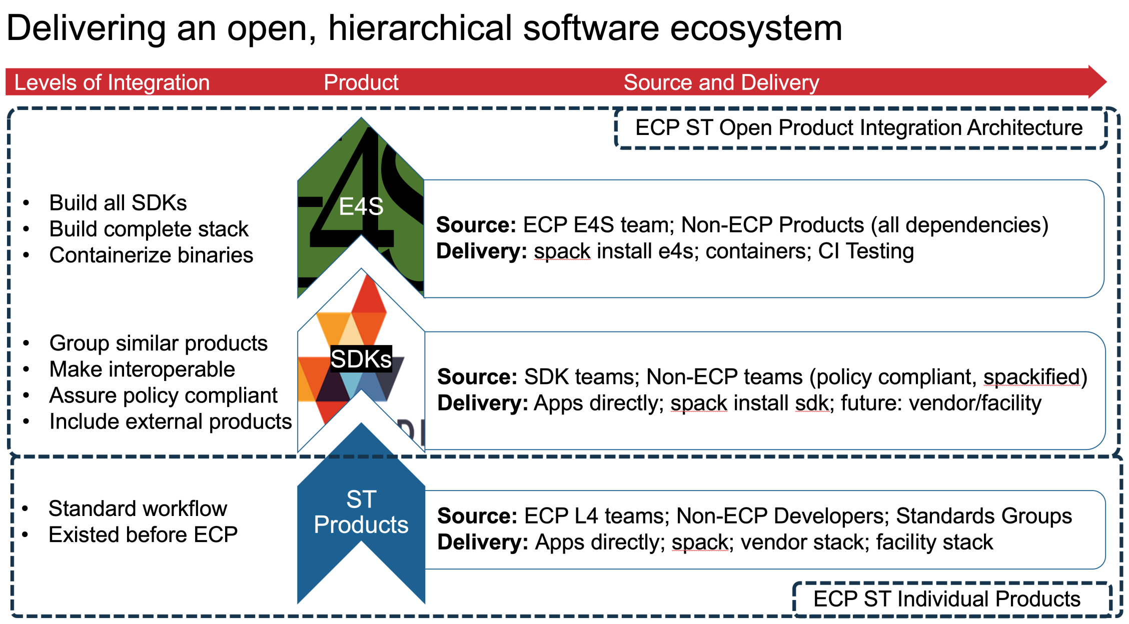
ECP ST Delivery Strategy ECP ST delivers it software products as source code, primarily in repositories found on GitHub, Gitlab installations or similar platforms. Clients such as ECP HI, OpenHPC and application developers with direct repository access then take the source and build, install and test our software. The delivery strategy is outlined in Figure 20.

Users access ECP ST products using these basic mechanisms:

- **Build from source code:** The vast majority of ECP ST products reach at least some of their user base via direct source code download from the product repository. In some cases, the user will download a single compressed file containing product source, then expand the file to expose the collection of source and build files. Increasingly, users will fork a new copy of an online repository. After obtaining the source, the user executes a configuration process that detects local compilers and libraries and then builds the product. This kind of access can represent a barrier for some users, since the user needs to build the product and can encounter a variety of challenges in that process, such as an incompatible compiler or a missing third-party library that must first be installed. However, building from source can be a preferred approach for users who want control over compiler settings, or want to adapt how the product is used, for example, turning on or off optional features, or creating adaptations that extend product capabilities. For example, large library frameworks such as PETSc and Trilinos have many tunable features that can benefit from the user building from source code. Furthermore, these frameworks support user-defined functional extensions that are easier to support when the user builds the product from source. ECP ST is leveraging and contributing to the development of Spack [1]. Via meta-data stored in a Spack *package* defined for each product, Spack leverages a product's native build environment, along with knowledge about its dependencies, to build the product and dependencies from source. Spack plays a central role in ECP ST software development and delivery processes by supporting turnkey builds of the ECP ST software stack for the purposes of continuous integration testing, installation and seamless multi-product builds.

- **DOE computing facilities:** Each DOE computing facility (ALCF, OLCF, NERSC, LLNL and ACES [LANL/SNL]) provides pre-built versions of 17 to 20 ECP ST products (although the exact mix of products varies somewhat at each site). Many of these products are what users would consider to be part of the core system capabilities, including compilers, e.g., Flang (Section 4.2.22) and LLVM (Section 4.2.19), and parallel programming environments such as MPICH (Section 4.1.8), OpenMPI (Section 4.1.14) and OpenMP (Section 4.2.21). Development tools such as PAPI (Section 4.2.8) and TAU (Section 4.2.17) are often part of this suite, if not already included in the vendor stack. Math and data libraries such as PETSc (Section 4.3.9), Trilinos (Section 4.3.17), HDF5 (Section 4.4.13) and others are also available in some facilities software installations. We anticipate and hope for increased collaboration with facilities via the ECP Hardware & Integration (HI) Focus Area. We are also encouraged by multi-lab efforts such as the Tri-Lab Operating System Stack (TOSS) [9] that are focused on improving uniformity of software stacks across facilities.

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



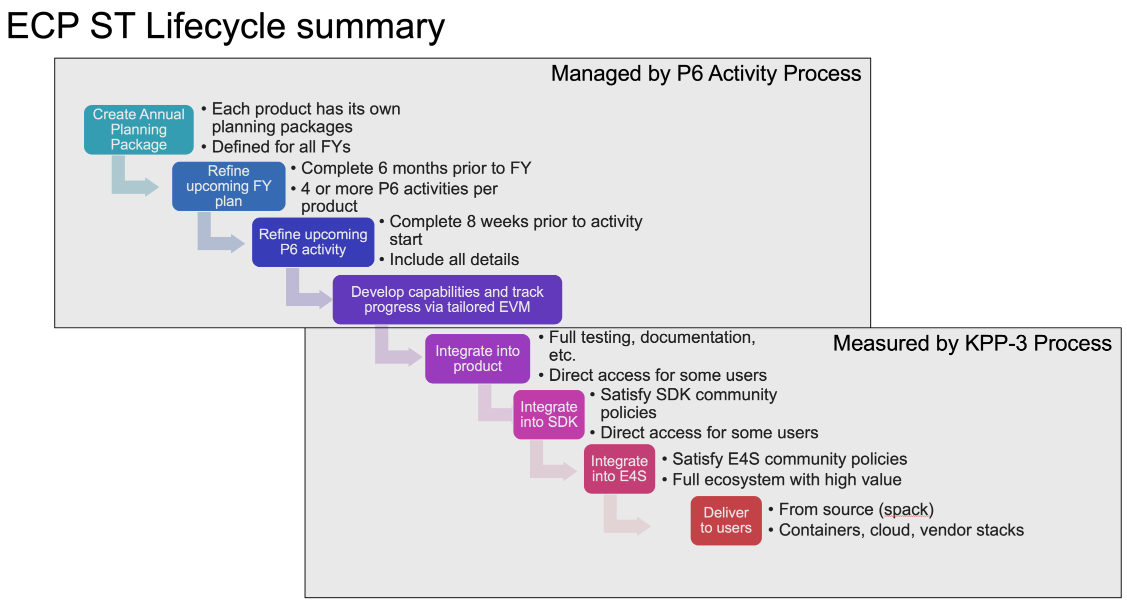
Delivering an open, hierarchical software ecosystem
Levels of Integration
Product
Source and Delivery
ECP ST Open Product Integration Architecture
Build all SDKs
Build complete stack
Containerize binaries
E4S
Source: ECP E4S team; Non-ECP Products (all dependencies)
Delivery: spack install e4s; containers; CI Testing
Group similar products
Make interoperable
Assure policy compliant
Include external products
SDKs
Source: SDK teams; Non-ECP teams (policy compliant, spackified)
Delivery: Apps directly; spack install sdk; future: vendor/facility
Standard workflow
Existed before ECP
ST Products
Source: ECP L4 teams; Non-ECP Developers; Standards Groups
Delivery: Apps directly; spack; vendor stack; facility stack
ECP ST Individual Products

Figure 20: The ECP ST software stack is delivered to the user community through several channels. Key channels are via source code, increasingly using SDKs, direct to Facilities in collaboration with ECP HI, via binary distributions from containers and HPC vendors. Increasingly, E4S is the primary pathway for delivering ECP ST capabilities. E4S provides testing, a documentation portal and quality commitments via community policies.

- **Binary distributions:** Approximately 10 ECP ST products are available via binary distributions such as common Linux distributions, in particular via OpenHPC[10]. ECP ST intends to foster growth of availability via binary distributions as an important way to increase the size of the user community and improve product sustainability via this broader user base.
- **Container and Cloud environments:** E4S is available via an increasing number of container and cloud environments (Docker, Shifter, Singularity and CharlieCloud, AWS and Google Cloud)¿

2.2.4 ECP ST Software Lifecycle

We complete Section 2 with a discussion of the ECP ST Software Lifecycle as shown in Figure 21. From inception as P6 Activity planning packages that are refined annually and given detailed information just prior to starting the activity to the successful integration of a capability into the client environment, ECP ST features are governed by this lifecycle. Of course, each product team conducts its own integrated planning that incorporates other funding sources and stakeholders, but the ECP ST Lifecycle intersects the product lifecycle for capabilities that ECP funds.



ECP ST Lifecycle summary
Managed by P6 Activity Process
Create Annual Planning Package
• Each product has its own planning packages
• Defined for all FYs
Refine upcoming FY plan
• Complete 6 months prior to FY
• 4 or more P6 activities per product
Refine upcoming P6 activity
• Complete 8 weeks prior to activity start
• Include all details
Develop capabilities and track progress via tailored EVM
Measured by KPP-3 Process
Integrate into product
• Full testing, documentation, etc.
• Direct access for some users
Integrate into SDK
• Satisfy SDK community policies
• Direct access for some users
Integrate into E4S
• Satisfy E4S community policies
• Full ecosystem with high value
Deliver to users
• From source (spack)
• Containers, cloud, vendor stacks

Figure 21: ECP ST product planning, executing, testing and assessment are governed by the combination of P6 Activities for hierarchical planning (Figure 18) and KPP-3 for measuring capability integrations (Table 3). This figure shows how the entire lifecycle of ECP ST feature development is captured by these two elements.

3. ECP ST DELIVERABLES

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

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

3.1 ECP ST DEVELOPMENT PROJECTS

ECP ST efforts support development in the following software projects in five technical areas (Table 1). In each table is a list of related projects, a URL (if available) and an estimate of deployment scope.

3.2 STANDARDS COMMITTEES

An important activity for ECP ST staff is participation in standards efforts. In many instances, our software will not be sustainable if it is not tightly connected to a standard. At the same time, any standard has to take into account the emerging requirements that Exascale platforms need in order to achieve performance and portability. Figure 22 summarized ECP ST staff involvement in the major standards efforts that impact ECP.

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

Table 6: Programming Models and Runtimes Projects (18 total).

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

Table 7: Development Tools Projects (22 total).

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

Table 8: Mathematical Libraries Projects (18 total).

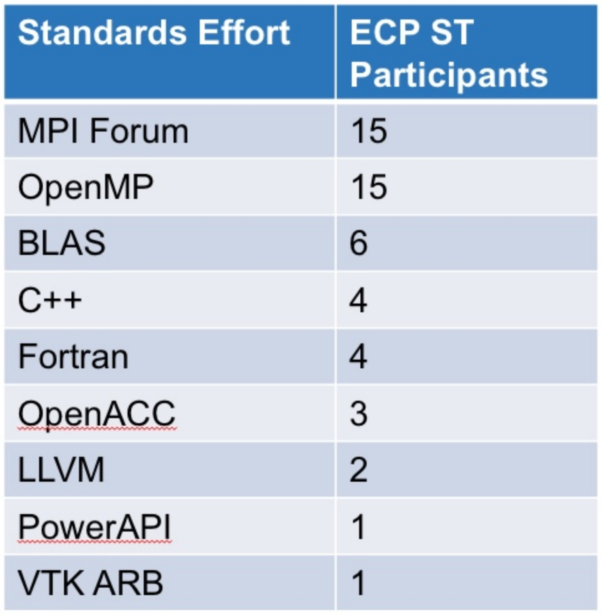
Product	**Website**	**Deployment Scope**
Catalyst (ALPINE)	https://www.paraview.org/in-situ	Broad
Darshan	http://www.mcs.anl.gov/research/projects/darshan	Broad
HDF5	https://www.hdfgroup.org/downloads	Broad
IOSS	https://github.com/gsjaardema/seacas	Broad
Parallel netCDF	http://cucis.ece.northwestern.edu/projects/PnetCDF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

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

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

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

ECP ST staff are heavily involved in MPI and OpenMP standards efforts. ECP ST staff hold several key leadership positions and have heavy involvement in all aspects. ECP ST staff also play a critical role in C++ standards efforts. While DOE staff have only recently engaged in C++ standards, our efforts are essential to getting HPC requirements considered, especially by contributing working code that demonstrates requirements and design. ECP ST sponsors the newest open source Fortran compiler Flang 4.2.22, a front end for LLVM. This compiler is a rapidly emerging and essential part of the HPC ecosystem. In particular, while ARM processors are not explicitly part of the pre-Exascale ecosystem, they are emerging as a strong contender in the future. Flang is *the* Fortran compiler for ARM-based systems. ECP ST involvement in other committees, including the *de facto* also provide valuable leverage and improved uniformity for HPC software. Lastly, we mention the Visualization Toolkit (VTK) Architecture Review Board (ARB). While this is only a single instance, we intend to explore the ARB model as part of our SDK efforts.



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

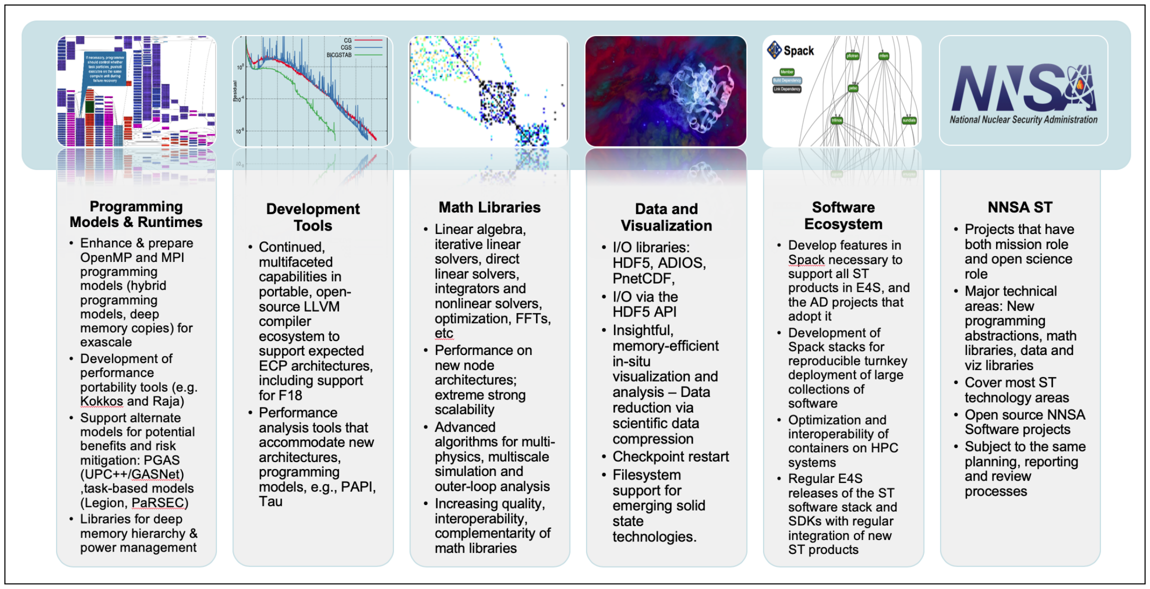
Figure 22: ECP ST staff are involved in a variety of official and *de facto* standards committees. Involvement in standards efforts is essential to assuring the sustainability of our products and to assure that emerging Exascale requirements are addressed by these standards.

3.3 CONTRIBUTIONS TO EXTERNAL SOFTWARE PRODUCTS

While much of ECP ST efforts and focus are on the product that we develop and support, it is important to note that some of our important work, and certainly some of our most sustainable and highly leveraged work, is done by providing requirements, analysis, design and prototype capabilities for vendor and other third party software. Many software studies have shown that 70 to 80% of the cost of a successful software product goes into post-delivery maintenance. Our effort summarized in Table 11 expressly eliminate this large cost for DOE because the product is developed and supported outside of DOE.

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

Table 11: External products to which ECP ST activities contribute. Participation in requirements, analysis, design and prototyping activities for third-party products is some of the most effective software work we can do.



Programming Models & Runtimes
- Enhance & prepare OpenMP and MPI programming models (hybrid programming models, deep memory copies) for exascale
- Development of performance portability tools (e.g. Kokkos and Raja)
- Support alternate models for potential benefits and risk mitigation: PGAS (UPC++/GASNet) ,task-based models (Legion, PaRSEC)
- Libraries for deep memory hierarchy & power management

Development Tools
- Continued, multifaceted capabilities in portable, open-source LLVM compiler ecosystem to support expected ECP architectures, including support for F18
- Performance analysis tools that accommodate new architectures, programming models, e.g., PAPI, Tau

Math Libraries
- Linear algebra, iterative linear solvers, direct linear solvers, integrators and nonlinear solvers, optimization, FFTs, etc
- Performance on new node architectures; extreme strong scalability
- Advanced algorithms for multi-physics, multiscale simulation and outer-loop analysis
- Increasing quality, interoperability, complementarity of math libraries

Data and Visualization
- I/O libraries: HDF5, ADIOS, PnetCDF,
- I/O via the HDF5 API
- Insightful, memory-efficient in-situ visualization and analysis – Data reduction via scientific data compression
- Checkpoint restart
- Filesystem support for emerging solid state technologies.

Software Ecosystem
- Develop features in Spack necessary to support all ST products in E4S, and the AD projects that adopt it
- Development of Spack stacks for reproducible turnkey deployment of large collections of software
- Optimization and interoperability of containers on HPC systems
- Regular E4S releases of the ST software stack and SDKs with regular integration of new ST products

NNSA ST
- Projects that have both mission role and open science role
- Major technical areas: New programming abstractions, math libraries, data and viz libraries
- Cover most ST technology areas
- Open source NNSA Software projects
- Subject to the same planning, reporting and review processes

Figure 23: ECP ST is composed of 6 Level-3 Technical Areas. The first four areas are organized around functionality development themes. The fifth is focused on technology for packaging and delivery of capabilities. The sixth is organized around per-lab open source development at the three DOE NNSA laboratories, LANL, LLNL and SNL.

4. ECP ST TECHNICAL AREAS

ECP ST is composed of six Level-3 Technical Areas (see Figure 23). In this section of the ECP ST Capabilities Assessment Report we provide an overview of each Level-3 Technical Area and two-page summaries of each funded project within the technical area. For each L3 area, we discuss scope and requirements, assumptions and feasibility, objectives, plans, risks and mitigations and future trends. For each Level-4 subproject, we provide a project overview and summarizes the key challenges, solution strategy, recent progress and next steps for the project.

4.1 WBS 2.3.1 PROGRAMMING MODELS & RUNTIMES

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

4.1.1 Scope and Requirements

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

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

4.1.2 Assumptions and Feasibility

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

4.1.3 Objectives

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

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

4.1.4 Plan

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

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

4.1.5 Risks and Mitigation Strategies

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

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

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

4.1.6 Future Trends

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

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

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

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

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

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

4.1.7 WBS 2.3.1.01 Programming Models & Runtimes Software Development Kits

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

4.1.8 WBS 2.3.1.07 *Exascale MPI*

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

MPICH team is involved in the formation of the MPI Forum and have been deeply involved in defining the MPI standard since 1992. MPICH has helped prototype and define the majority of the features in the MPI standard. As such, MPICH has been one of the most influential pieces of software in accelerating the adoption of the MPI standard by the HPC community. MPICH has been adopted by leading vendors into their own derivative implementations. Examples include Intel (for Intel MPI), Cray (for Cray MPI), IBM (for IBM PE MPI), Mellanox (for MLNX-MPI), Microsoft (for MS-MPI), and Ohio State University (for MVAPICH). MPICH and its derivatives are exclusively used in 7 of the top 10 supercomputers in the world today. MPICH is the recipient of a number of awards including an R&D 100 award.

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

1. Interoperability with intranode programming models having a high thread count [11, 12, 13] (such as OpenMP, OpenACC and emerging asynchronous task models);
2. Scalability and performance over complex architectures [14, 15, 13, 16] (including high core counts, processor heterogeneity and heterogeneous memory);
3. Software overheads that are exacerbated by lightweight cores and low-latency networks;
4. Enhanced functionality (extensions to the MPI standard) based on experience with applications and high-level libraries/frameworks targeted at Exascale; and
5. Topics that become more significant as we move to the next generation of HPC architectures: memory usage, power, and resilience.

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

Our solution strategy started by addressing performance and scalability aspects in MPICH related to network address management [17]. Apart from this, we also looked at communication strategies which allow the MPI library to be as lightweight as possible [18, 19]. Other solutions include investigation and evaluation of communication relaxation hints, investigation of optimizations to memory scalability in MPICH and improvements to MPI RMA operations.

Exascale MPI heterogeneity efforts [20, 21, 22] started with the survey on heterogeneous memory architectures on upcoming DOE machines and how MPICH can take advantage of them [23]. The efforts also included the investigation of utilizing heterogeneous memory inside the MPI implementation and evaluation of applications [24]. The heterogeneity efforts further extended to investigating and developing technologies for GPU integration for the better support of the coming Exascale supercomputers.

Exascale MPI topology awareness efforts [25, 26] originated with the investigation and evaluation of hints based on topology awareness and optimizations to virtual topology functionality in MPICH [27, 28]. The other efforts include investigation of topology-aware collectives and neighborhood collectives in MPICH [29] and evaluation of the selected ECP applications.

Exascale MPI fault tolerance efforts [30, 13] started with support for handling noncatastrophic errors in MPI. The second effort included defining the scope of errors in MPI, a prerequisite for user-level failure mitigation. Other efforts in this direction includes standardizing these efforts in MPI and evaluating application suitability for fault tolerance.

Exascale MPI+X hybrid programming developed firstly with effort in improving interoperation of MPICH with threads [31]. Secondly, we developed the work-queue data transfer model for multithreaded MPI communication [32]. We have included support for interaction of MPICH with user-level thread (ULT) libraries [33], primarily targeting Argobots and the BOLT runtime [34]. Other issues that are being looked at include the investigation and evaluation on multiple virtual communication interfaces for multithreaded MPI communication [35].

Recent Progress The recent release of MPICH 3.4b1 version includes deliverables from the major milestones of FY2020. Figure 24 provides the details of them. In the first milestone, we created the collective algorithm selection framework which allows MPI to choose different algorithm for collectives operations based on multiple factors including number of processes, network topology and support of hardware acceleration. The user of MPICH can provide a configuration file to describe the criteria for choosing different collective algorithms. We expect the vendors utilizing this capability to provide collective selection strategies that is optimized for the target supercomputers. The end-user could also generate their own strategies that is fine-tuned to the characteristic and needs of their applications. In the second milestone, we added the support for GPU in MPI communication. This allows a buffer in the GPU memory to be used directly in MPI communication operations. MPICH's GPU support can utilize the native support of fast GPU communication in hardware while also provide a fallback mode for more complex MPI communication pattens. We also created a new high-performance datatype engine that further utilizes GPU for handling non-contiguous datatype in MPI communication.

Lastly, we made many efforts in participating the standardization of MPI-4 by discussing proposed features and contributing the feedback for our users back to the MPI forum. This has led to several new features being accepted by the MPI Forum.

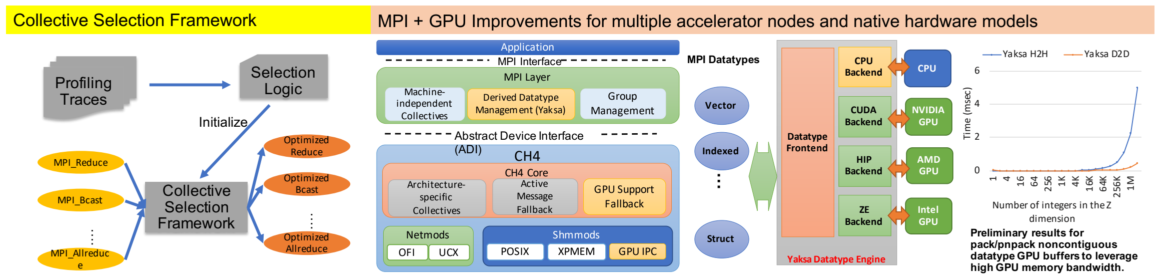


Figure 24: Major MPICH milestones completed in fiscal year 2020

Next Steps A major focus of the ongoing Exascale MPI efforts is further improvements to the MPI+GPU support. This includes support for GPU stream triggered MPI operations, optimized support for noncontiguous data and software evaluations. Another focus of FY2021 is the implementation of new features in MPI-4 standard including persistent collective operations. We will also make effort for improving the MPICH testing infrastructure by improving the testsuite and promoting integration ECP systems to provide better test coverage and a close-to-production test environment. This would enhance the readiness of MPICH for exascale systems.

4.1.9 WBS 2.3.1.08 Legion

Overview This project focuses on the development, hardening and support of the Legion Programming System (https://legion.stanford.edu) with a focus on Exascale platforms and workloads that can benefit from Legion's features. At a fundamental level our focus is on the key capabilities (e.g. correctness, performance, scalability) of an alternative programming model for ECP that seeks to expose additional levels of parallelism in applications. In addition, Legion also enables a separation of concerns of the implementation of an application from how it is mapped onto a given system architecture.

Our efforts are currently focused on addressing bugs, refactoring the implementation for improved stability, performance and scaling, extending support for the selected exascale platforms (Aurora and Frontier), and also expanding the feature set as needed for both application and platform nuances.

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

https://gitlab.com/StanfordLegion/legion

Key Challenges While Legion addresses a number of key challenges in improving system utilization, some aspects of platform portability, and is becoming more widely used, it is still a new programming system and therefore there is a cost to rewriting applications. This aspect makes significant adoption a risk within ECP and additional effort is being taken to improve stability and find unique use cases. aspects of performance and scaling to match aspects of more mature technologies.

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

Solution Strategy As a collaboration between Los Alamos, Stanford University and R&D efforts at NVIDIA, SLAC, Facebook, MIT, and others. We are providing the overarching implementation of Legion that captures the most stable (correct and feature complete) version of the programming system. In addition, we are actively looking for opportunities to educate the community about Legion and other data-centric and task-based approaches to programming.

We have continued working with ExaFEL (AD 2.2.4.05) and the CANDLE project (AD 2.2.4.03) to provide support for Legion. We also provide support and software releases related to the efforts going on within LANL's ATDM Programming Models and Runtimes project (part of ST 2.3.6.01), that refine, identify needs and requirements that are in support of Ristra (LANL's National Security application AD 2.2.5.01). Our project includes management of the current repository and quarterly, or more frequent, releases of Legion to the broader community. We are also supporting approaches that support Legion inter-operation with other languages and programming systems – e.g. MPI, OpenMP, Kokkos, Fortran, Python.

We have continued our work on improving performance and scaling of training deep learning applications. In particular, we are working closely with CANDLE's requirements for ML training and inference on large DNNs. Our most recent progress is discussed below.

Recent Progress We continue to discover and address both performance and scalability issues in the runtime. In addition, for use cases within ECP, and also a growing set of users outside of ECP, we have continued to identify and address bugs and other issues (e.g. missing features).

As mentioned above, our we continue to focus on improving training times for CANDLE's DNN use cases and also improving developer productivity when using Flexflow (the DNN training layer built on top of Legion). We are specifically working to make sure the feature set of FlexFlow system provides the necessary functionality and as part of this we have recently completed a Python layer for FlexFlow that provides support for the Keras (TensorFlow) interface. This allows TensorFlow programs to be run using FlexFlow with very minimal changes to the original Python code. FlexFlow is now available as open source under an Apache License:

https://github.com/flexflow/FlexFlow

We are now starting work on new training networks with CANDLE and also working on providing a PyTorch compatibility interface to provide the same ability to run widely used Python applications for machine learning using Flow and Legion as the underlying runtime systems.

Finally, we currently have a start on supporting both AMD and Intel GPUs. After AMD deprecated the CUDA Driver API in their HIP software layer we had to rewrite a portion of the underlying implementation to re-establish AMD support. This initial work is complete and Legion applications are running successfully on currently available AMD GPUs. Similar efforts are underway for Intel systems using the oneAPI interface. The initial implementation has run into a few issues within Intel's software stack that we are waiting to be resolved. We are on target to successfully run on Intel's stack by late in 2020 or early 2021. We have also completed an additional lower-level MPI-based transport layer underneath Legion to provide a level of risk mitigation at the lower levels of the Legion runtime (Realm).

Next Steps Our plans for the next year are to continue focusing on the challenges presented by the upcoming exascale system architectures and on hardening and improving the overall performance and scalability of Legion. These efforts will specifically begin to focus on AMD and Intel hardware with an eye towards performance at the node level (limited by the availability and stability of appropriate hardware resources).

We will continue to work on the Python interfaces for Legion with a focus on the Keras and PyTorch features requested by CANDLE. This will include seeking out and improving our outreach. Our regular open source releases of Legion and FlexFlow will continue. Further work will need to be done to focus on bug fixes, improving capabilities, improving developer productivity, and addressing performance issues on both existing and upcoming platforms.

4.1.10 WBS 2.3.1.09 Distributed Tasking at Exascale: PaRSEC

Overview The PaRSEC Environment provides a software ecosystem composed of a runtime component to dynamically execute task-based applications on heterogeneous distributed systems, and a productivity toolbox that comprises a development framework for the support of multiple Domain Specific Languages (DSLs) and extensions, with debugging, trace collection, and analysis tools. The PaRSEC project team is dedicated to solving two challenging and interdependent problems facing the ECP developer community: First, how to create an execution model that enables developers to express as much parallelism as possible in their applications, so that applications effectively utilize the massive collection of heterogeneous devices ECP machines will deploy. Second, how to ensure the execution model is flexible and portable enough to actually provide and sustain a performance benefit by increasing the scientific productivity of the application developers, not only for the ECP target environments but for the foreseeable future.

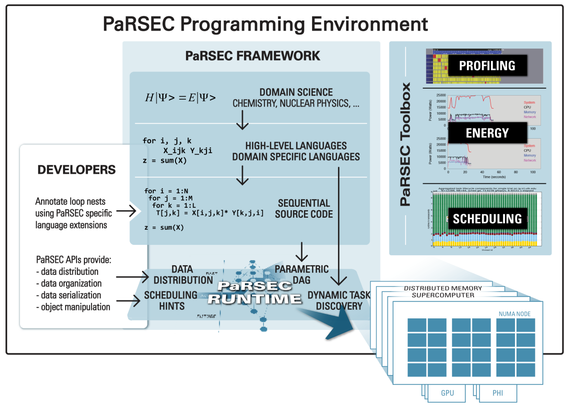


Figure 25: PaRSEC architecture based on a modular framework where each component can be dynamically activated as needed.

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

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

Solution Strategy The approach followed in PaRSEC is to provide a low-level, flexible and dynamic runtime able not only to schedule tasks at the node level, but to handle data transfers between different memory (both inter- and intra-nodes), memory hierarchies, heterogeneous architectures with support for accelerators with a simple programming scheme. The proposed approach envisions a middle-ground solution, addressing both hardware and software challenges. At the hardware level a team of dedicated developers extends PaRSEC to map its capabilities to the hardware and to improve its scalability and performance. At the upper software level the runtime interactions are through Domain Specific Languages with the target domain scientists in mind, that will facilitate the expression of algorithmic parallelism with familiar constructs mapped on the exposed low-level capabilities. To facilitate the integration of PaRSEC-driven libraries into larger and complex applications, PaRSEC natively interoperate with other programming paradigms, including some target of the ECP PMR support, such as PGAS, MPI, OpenMP and Kokkos. This integration provides a smooth transition for library developers that embrace the PaRSEC runtime, providing a platform where a shift to a new programming paradigms can be done in stages of increased complexity [36, 37, 38].

Recent Progress The software release (2020.10) provides many new additions to the low-level task runtime, supports for a number of hardware capabilities (multi-stream GPU, NVLink, P9 atomic ops, ARM SVE support), brings significant improvements to the performance and scalability of the runtime, and addresses many pending issues. On the software quality side, the PaRSEC runtime has been evaluated and amended to compile and run on all pre-Exascale platforms (ALCF Mira, Theta; OLCF Summit), as well as some early platforms based on the new ARM architecture. PaRSEC now includes a Spack definition file to ease the deployment on future target systems as part of the system software xSDK effort.

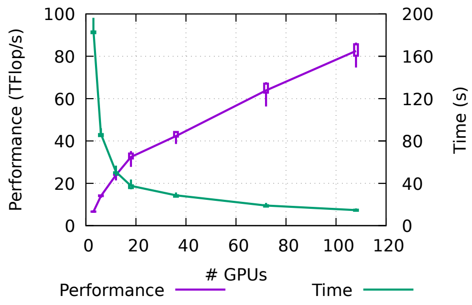


Figure 26: Time to solution and Performance as a function of the number of V100 GPUs on Summit, for the molecule $C_{65}H_{132}$

The PaRSEC team, in collaboration with NWChemEx project researchers, developed an efficient and portable tensor product algorithm specifically designed for the computational chemistry domain needs on top of the PaRSEC runtime. This includes an efficient matrix product operation for hybrid architectures, like Summit, with an irregularly blocked, block-sparse representation of matrices. Moreover, the requirements on this implementation were extremely strict, the matrices are rectangular and extremely large in at least one of their dimensions, such that none of the input matrices could fit in the aggregated memory of all GPUs. The algorithm and a deeper analysis of the results are described in [39].

Figure 26 shows a strong scaling performance evaluation of this algorithm, when applied to the main tensor product required by the CCSD method to simulate the electronic structure from first principles. The simulated molecule was $C_{65}H_{132}$, which is a quasi-1-dimensional system and small atomic orbital basis, where the sparsity of tensors is maximized while the optimal (from the data compression perspective) tile size is small. That molecule is representative of applications to 1-d polymers and quasi-linear molecules (such as some proteins), while the choice of the atomic orbital basis is representative of medium-precision simulations in chemistry and condensed phase. Such use case stresses the system and algorithm as it implies a significant sparsity: the largest matrix, while being of square of rank $2,464,900$, has only a density of 3.1%.

The strong scaling evaluation shows a parallel efficiency of 35%, with a time to solution at 180s on 3 GPUs, going down to 13s at 118 GPUs. Compared to the state of the art, DBCSR [40] can only run problems that fit in the GPU memory, preventing us to run the same experiment, but experiments on synthetic problems that fit in memory show an improvement by a factor two, while TiledArray [41] cannot leverage the GPUs of Summit without using PaRSEC and would run, on the CPUs of Summit ten times slower.

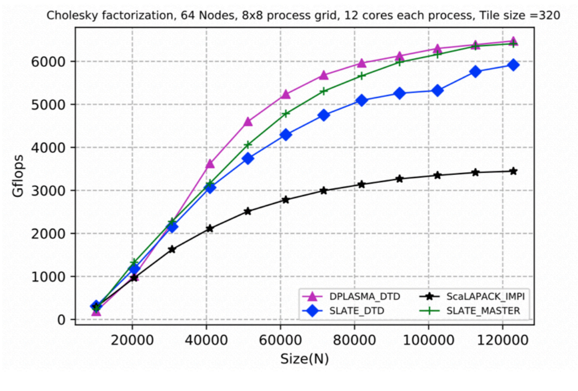
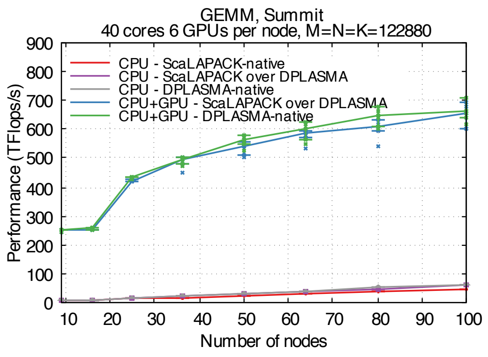


Figure 27: Comparison of DPLASMA and SLATE Cholesky factorization over PaRSEC with SLATE and ScaLAPACK on 64 nodes 12 cores each

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

In this context it is interesting to highlight the first step toward the integration of the PaRSEC framework into the SLATE (2.3.3.09) in the context of the shared milestone (STPM11-23). The first prototype of the application ran in a distributed environment and showed the capability of the SLATE library using a modern fully capable runtime system. This work involved enhancing the insert task interface available in the ParSEC runtime to map onto the logic of a SLATE algorithm.

Figure 27 compares different implementation of the Cholesky factorization. On one side we have two reference implementation for distributed linear algebra, ScaLAPACK and the current version of the SLATE library (using OpenMP for intra-node parallelism and MPI for communications). On the other side we have two DSL expressing the same algorithm but using PaRSEC as the underlying runtime, the Dynamic Task Discovery (DTD) an approach similar to OpenMP but working on a distributed setting, and a version of the SLATE library where instead of relying on explicit parallelism (OpenMP) and communications (MPI) it rely on implicit dependencies management via PaRSEC.



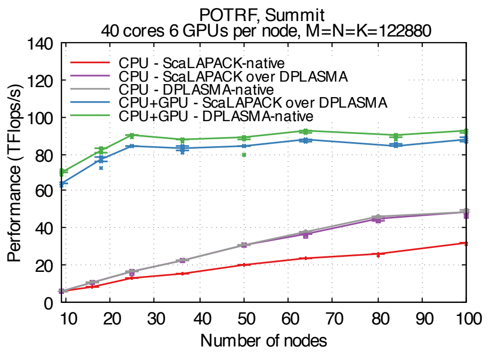


Figure 28: Performance using GPUs of native ScaLAPACK, ScaLAPACK over DPLASMA and native DPLASMA for GEMM and POTRF.

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

Figure 28 compare the performance of the ScaLAPACK wrapper extensions against the native DPLASMA and native ScaLAPACK in their typical usage scenarios (one process per core for ScaLAPACK), while DPLASMA tests (native DPLASMA and ScaLAPACK over DPLASMA) use one thread per core and one process per node on the experiments using only the CPU and two processes per node, each exploiting 3 GPUs, are used on the GPU tests. In all cases, the performance achieved by running ScaLAPACK over DPLASMA is more than 90% the performance of native DPLASMA. For the CPU-only experiments, the usage of the DPLASMA wrapper introduces a speedup of 1.73x for GEMM (minimum 1.64x, maximum 1.88x) and 1.49x for POTRF (minimum 1.27x, maximum 1.71x) . When exploiting also the GPUs of the computation nodes, the performance is increased by 16.75x for GEMM (minimum 10.79x, maximum 25.68x) and by 4.27x for POTRF (minimum 2.77x, maximum 6.70x). The lower performance of the DPLASMA's POTRF over GPUs is explained because in the current implementation of this algorithm not all the tasks are run on GPUs. However, further improvements of DPLASMA algorithms that will be integrated in the next release of DPLASMA will likely achieved similar performance when used for wrapping the ScaLAPACK routines. Therefore, enabling applications already using the ScaLAPACK API to improve their usage of the available hardware resource with very little effort that is translated in important performance benefits.

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

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

4.1.11 WBS 2.3.1.14 GASNet-EX

Overview The Lightweight Communication and Global Address Space Support project (Pagoda) is developing GASNet-EX [42], a portable high-performance communication layer supporting multiple implementations of the Partitioned Global Address Space (PGAS) model. GASNet-EX clients include Pagoda's PGAS programming interface UPC++ [43, 44] and the Legion Programming System [45, 46] (WBS 2.3.1.08).

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

We are co-designing GASNet-EX with the UPC++ development team with additional input from the Legion and (non-ECP) Cray Chapel [47, 48] projects.

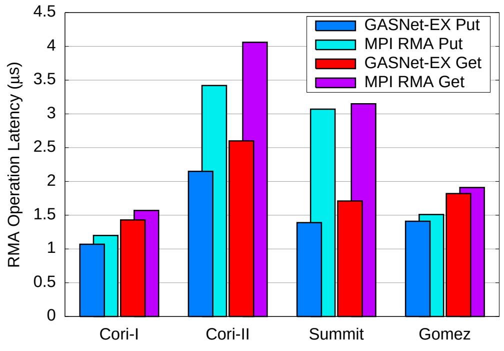
Key Challenges Exascale systems will deliver exponential growth in on-chip parallelism and reduced memory capacity per core, increasing the importance of strong scaling and finer-grained communication events. The pervasive use of accelerators introduces heterogeneous systems in which the engines providing the majority of the compute capability are not well suited to other tasks. Success at exascale demands that software minimize the overheads incurred upon lightweight cores and accelerators, especially avoiding long, branchy serial code paths; this motivates a requirement for efficient fine-grained communication. These problems are exacerbated by application trends; many of the ECP applications require adaptive meshes, sparse matrices, or dynamic load balancing. All of these characteristics favor the use of low-overhead communication mechanisms that can maximize injection rate and network utilization, tolerate latency through overlap, accommodate unpredictable communication events, minimize synchronization, leverage hardware support for communication involving accelerator memory, and efficiently support small- to medium-sized messages. The ECP software stack needs to expose the best-available communication mechanisms, including novel features being developed by the vendor community.

Solution Strategy The PGAS model is a powerful means of addressing these challenges and is critical in building other ECP programming systems, libraries, and applications. We use the term *PGAS* for models that support one-sided communication, including contiguous and non-contiguous remote memory access (RMA) operations such as put/get and atomic updates. Some of these models also include support for remote function invocation. GASNet-EX [49] is a communications library that provides the foundation for implementing PGAS models, and is the successor to the widely-deployed GASNet library. We are building on over 15 years of experience with the GASNet [42, 50] communication layer to provide production-quality implementations that include improvements motivated by technology trends and application experience.

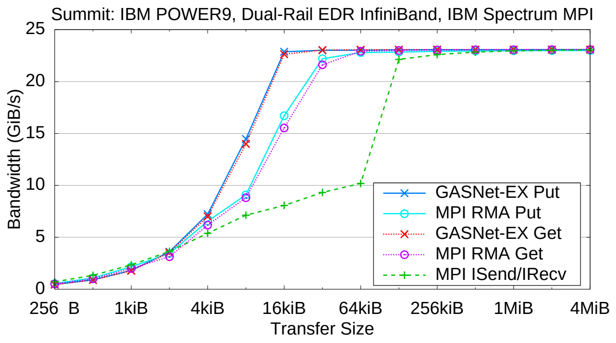
The goal of the GASNet-EX team is to provide a portable, high-performance PGAS communication layer for exascale and pre-exascale systems, addressing the challenges identified above. GASNet-EX provides interfaces that efficiently match the RDMA capabilities of modern inter-node network hardware and intra-node communication between distinct address spaces. New interfaces for atomics and collectives have enabled offload to current and future network hardware with corresponding capabilities. These design choices and their implementations supply the low-overhead communication mechanisms required to address the requirements of exascale applications.

Figure 29 shows representative results from a paper [49] comparing the RMA performance of GASNet-EX against MPI on multiple systems including NERSC's Cori and OLCF's Summit[1]. These results demonstrate the ability of a PGAS-centric runtime to deliver performance as good as MPI, and often better. The paper presents experimental methodology and system descriptions, which are also available online [42], along with results for additional systems.

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



(a) 8-byte RMA Latencies



(b) Summit Flood Bandwidth

Figure 29: Selected GASNet-EX vs. MPI RMA Performance Results

Figure 29a shows the latency of 8-byte RMA Put and Get operations on four systems, including two distinct network types and three distinct MPI implementations. GASNet-EX's latency is 6% to 55% better than MPI's on Put and 5% to 45% better on Get. Algorithms sensitive to small-transfer latency may become practical in PGAS programming models due to these improvements relative to MPI.

Figure 29b shows flood bandwidth of RMA Put and Get over the dual-rail InfiniBand network of OLCF's Summit. GASNet-EX's bandwidth is seen to rise to saturation at smaller transfer sizes than IBM Spectrum MPI, with the most pronounced differences appearing between 4KiB and 32KiB. Comparison to the bandwidth of MPI message-passing (dashed green series) illustrates the benefits of one-sided communication, a major feature of PGAS models.

Recent Progress The most notable work on GASNet-EX in the past year has been in two areas:

Device (GPU) Memory Support. "Memory kinds" is the GASNet-EX term for support for communication involving memory other than host memory, and in the context of ECP refers primarily to accelerator devices such GPUs. The GASNet-EX APIs for memory kinds have been co-designed with the developers of UPC++ and the Realm runtime layer of the Legion Programming System (WBS 2.3.1.08). Starting in October 2020, GASNet-EX can now leverage the GPUDirect RDMA (GDR) capabilities of modern NVIDIA GPUs and Mellanox network adapters (such as those on Summit) to perform one-sided RMA involving GPU memory without the overheads of staging through intermediate buffers in host memory.

Scalability. We have devoted effort in the past year to reducing the memory footprint of the GASNet runtime as the job size grows. This has included efforts in collaboration with the ExaBiome (WBS 2.2.4.04) team to run their applications at previously unattainable scales on Summit at the OCLF and on Cori at NERSC.

Next Steps Our next efforts include:

Device (GPU) Memory Support. We will continue work in the area of GASNet-EX memory kinds, including the hardening and tuning of the implementation featured in the October 2020 release. As access to other ECP-relevant systems is secured, we plan to extend support to accelerators from additional vendors, including those from AMD and Intel which are scheduled to appear in early exascale systems.

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

4.1.12 WBS 2.3.1.14 UPC++

Overview The Lightweight Communication and Global Address Space Support project (Pagoda) is developing UPC++ [44], a C++ library supporting Partitioned Global Address Space (PGAS) programming [43, 51]. The current UPC++ v1.0 (distinct from an earlier prototype designated V0.1 [52]) is distinguished by three guiding principles. First, all communication is *asynchronous*, allowing overlap of computation and communication, and encouraging programmers to avoid global synchronization. Second, all communication is *syntactically explicit*, encouraging programmers to consider the costs of communication. Third, UPC++ encourages the use of *scalable data-structures*, avoiding non-scalable library features. These principles provide a programming model that can scale efficiently to potentially millions of processors. UPC++ is well-suited for implementing elaborate distributed data structures where communication is irregular or fine-grained. The UPC++ communication interfaces for Remote Memory Access (RMA) and Remote Procedure Calls (RPC) are composable and fit naturally within the context of modern C++.

UPC++ is needed for ECP because it delivers low-overhead communication, embracing interest by vendors in the PGAS model to efficiently match RDMA capabilities of modern network hardware and on-chip communication between distinct address spaces. Because ECP applications rely on irregular representations to improve accuracy and conserve memory, the UPC++ library provides an essential ingredient for the ECP software stack. It enables effective scaling of exascale software by reducing the work funneled to lightweight cores, avoiding the overhead of long, branchy serial code paths, and providing efficient fine-grained communication. The importance of these properties is reinforced by application trends; many ECP applications require the use of irregular data structures such as adaptive meshes, sparse matrices, particles, or similar techniques, and also perform load balancing. UPC++'s low-overhead communication mechanisms can maximize injection rate and network utilization, tolerate latency through overlap, streamline unpredictable communication events, minimize synchronization, leverage hardware support for communication involving accelerator memory, and efficiently support small- to medium-sized messages arising in such applications. UPC++ enables the ECP software stack to exploit the best-available communication mechanisms, including novel features being developed by vendors. UPC++ provides seamless and efficient multithreading support, offering a complementary and interoperable approach to "MPI + X", enabling developers to focus their effort on optimizing performance-critical communication.

Key Challenges As a result of technological trends, the cost of data motion is steadily increasing relative to that of computation. To reduce communication costs, we need to reduce the software overheads and hide communication latency behind available computation. UPC++ addresses both strategies. UPC++ avoids the software overheads associated with MPI, instead relying on the GASNet-EX [42, 49] communication library which is specifically designed and tuned for native PGAS communication using the best-available hardware mechanisms on each network (see Section 4.1.11 on GASNet-EX, which is being co-designed). UPC++ supports asynchronous communication via one-sided RMA and RPC.

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

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

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

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

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

As one example of how our approach is applicable to real problems we have implemented a distributed hash table, which serves as a proxy for a key phase in the HipMer application of the Exabiome Project

(WBS 2.2.4.04). This implementation scales efficiently to a large number of processors. RPC was observed to simplify the implementation considerably, by avoiding data hazards without the need for locking. Figure 30 illustrates the benefits of the UPC++ model in a weak scaling study up to 34,816 processes on the KNL partition of NERSC's Cori.

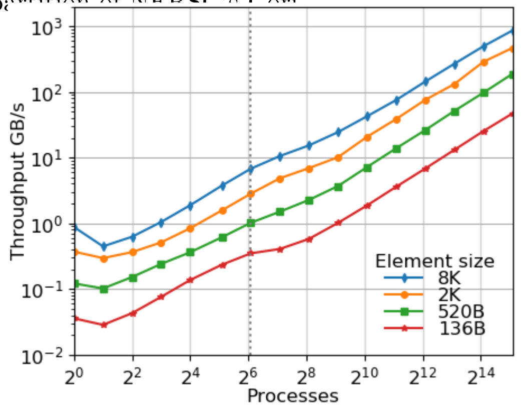


Figure 30: Weak scaling of distributed hash table insertion on the KNL partition of NERSC's Cori platform. The dotted line represents one node.

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

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

2. Training and Outreach. In the past year, the UPC++ team has increased focus on outreach. This includes presenting four training events, and preparation of a fifth to appear at SC20. A two-hour "Birds of a Feather" event in August 2020 introduced current and potential UPC++ users to the authors of successful UPC++-based applications. Circulation of working group drafts of proposed enhancements has been valuable to both the UPC++ team and our stakeholders to agree upon design in advance of implementation.

3. Productivity and Performance. Having completed the core specification and implementation of UPC++, we have shifted focus toward addressing improvements to productivity and performance, especially in response to stakeholder feedback. The most significant example is addition of support for non-trivial serialization of user-defined types, providing concise syntax for the common cases and robust, extensible mechanisms for more complex ones.

Next Steps The planned work for the near-term future includes the following:

1. Memory Kinds. We will continue development of the October 2020 memory kinds prototype. The implementation, currently targeting the hardware in Summit, will be extended to include other accelerators scheduled to appear in the early Exascale systems. As with Summit, transfers will be offloaded to available hardware capabilities by leveraging the parallel efforts in GASNet-EX.

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

3. Outreach. We will continue to hold training events for users of UPC++ and circulate working group drafts of productivity features (above) to solicit feedback.

4.1.13 WBS 2.3.1.16 SICM

Overview The goal of this project is to create a universal interface for discovering, managing and sharing within complex memory hierarchies. The result will be a memory API and a software library which implements it. These will allow operating system, runtime and application developers and vendors to access emerging memory technologies. The impact of the project will be immediate and potentially wide reaching, as developers in all areas are struggling to add support for the new memory technologies, each of which offers their own programming interface. The problem we are addressing is how to program the deluge of existing and emerging complex memory technologies on HPC systems. This includes the MCDRAM (on Intel Knights Landing), NV-DIMM, PCI-E NVM, SATA NVM, 3D stacked memory, PCM, memristor, and 3Dxpoint. Also, near node technologies, such as PCI-switch accessible memory or network attached memories, have been proposed in exascale memory designs. Current practice depends on ad hoc solutions rather than a uniform API that provides the needed specificity and portability. This approach is already insufficient and future memory technologies will only exacerbate the problem by adding additional proprietary APIs. Our solution is to provide a unified two-tier node-level complex memory API. The target for the low-level interface are system and runtime developers, as well as expert application developers that prefer full control of what memory types the application is using. The high-level interface is designed for application developers who would rather define coarser-level constraints on the types of memories the application needs and leave out the details of the memory management. The low-level interface is primarily an engineering and implementation project. The solution it provides is urgently needed by the HPC community; as developers work independently to support these novel memory technologies, time and effort is wasted on redundant solutions and overlapping implementations. Adoption of the software is focused on adsorption into existing open source projects such as hwloc, Umpire, CLANG/LLVM, OpenMP, and Jemalloc.

- Low-Level Interface: Finished refactor of low-level interface supporting memory arenas on different memory types. Added initial support for Umpire, OpenMP. Reviewing features need to fully support these runtimes. SICM now supports Intel Optane memory, the first NVM memory that can be used as an extension of traditional DRAM memory. Pull requests have been developed for OpenMP/CLANG/LLVM and Umpire. the patches to Clang/LLVM/OpenMP turn OpenMP memory spaces in OpenMP 5.x into SICM library calls in the LLVM/OpenMP runtime. The same codepath that supports memkind library was refactored to support multiple custom memory allocators – more general than just SICM support. SICM currently supports "pragma openmp allocate" with memory types: omp_ (default, large_cap, const, high_bw, low_lat) _mem_spaces and supports KNL, Optane, testing on Sierra/Summit.
- High-Level Graph Interface: Metall is a persistent memory allocator designed to provide developers with an API to allocate custom C++ data structures in both block-storage and byte- addressable persistent memories (e.g., NVMe and Intel Optane DC Persistent Memory) beyond a single process lifetime. Metall relies on a file-backed mmap mechanism to map a file in a filesystem into the virtual memory of an application, allowing the application to access the mapped region as if it were regular memory which can be larger than the physical main-memory of the system.
- Analysis: SICM has employed application profiling and analysis to direct data management across complex memory hierarchy, the team extended the SICM high-level interface with application-directed data tiering based on the MemBrain approach which is more effective than an unguided first touch policy. The impact of using different data features to steer hot program data into capacity-constrained device tiers was modeled.

Next Steps

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

- For the analysis work the team will extend and harden the tools for guiding application memory management and investigate feature categories to classify objects associated with different features such as size, type, allocation time, etc to guide data placement.

- For the Metall high-level interface we plan to continue outreach to ExaGraph to store graph data as well as other intermediate data into PM leveraging Metall. We also plan to support UMap (user-level mmap library in Argo PowerSteering project) underneath Metall to enhance its performance and capability.

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

Overview The OMPI-X project ensures that the Message Passing Interface (MPI) standard, and its specific implementation in Open MPI meet the needs of the ECP community in terms of performance, scalability, and capabilities or features. MPI is the predominant interface for inter-process communication in high-end computing. Nearly all of the ECP application (AD) projects (93% [53]) and the majority of software technology (ST) projects (57% [53]) rely on it. With the impending exascale era, the pace of change and growing diversity of HPC architectures pose new challenges that the MPI standard must address. The OMPI-X project is active in the MPI Forum standards organization, and works within it to raise and resolve key issues facing ECP applications and libraries.

Open MPI is an open source, community-based implementation of the MPI standard that is used by a number of prominent HPC vendors as the basis for their commercial MPI offerings. The OMPI-X team is comprised of active members of the Open MPI community, with an extensive history of contributions to this community. The OMPI-X project focuses on prototyping and demonstrating exascale-relevant proposals under consideration by the MPI Forum, as well as improving the fundamental performance and scalability of Open MPI, particularly for exascale-relevant platforms and job sizes. MPI users will be able to take advantage of these enhancements simply by linking against recent builds of the Open MPI library.

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

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

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

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

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

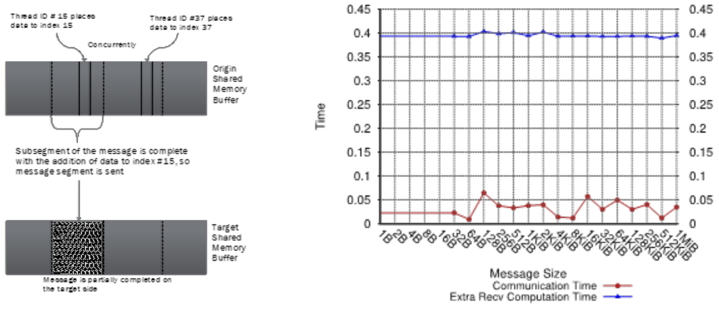

Thread ID # 15 places data to index 15
Thread ID #37 places data to index 37
Concurrently
Origin Shared Memory Buffer
Subsegment of the message is complete with the addition of data to index #15, so message segment is sent
Target Shared Memory Buffer
Message is partially completed on the target side
Time
Message Size
Communication Time
Extra Recv Computation Time
1B 2B 4B 8B 16B 32B 64B 128B 256B 512B 1KiB 2KiB 4KiB 8KiB 16KiB 32KiB 64KiB 128KiB 256KiB 512KiB 1MiB

Figure 31: Partitioned communications enables increased concurrency in communication operations which can be carried out by multiple threads or tasks.

into the MPI standard by developing prototypes and working with ECP teams and the broader community to demonstrate their feasibility and value.

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

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

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

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

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

Recent Progress Through extensive efforts with the MPI Forum community this year, OMPI-X has been successful in introducing a number of important exascale-related features into the forthcoming MPI 4.0 standard. These include sessions, partitioned communications, and a number of error management and recovery features based on the long-standing User-Level Fault Mitigation (ULFM) concept. These capabilities have at least prototype-level implementations available in the Open MPI library, allowing interested project teams to start exploring the new capabilities. We also undertook a cleanup of the Open MPI code base to improve the quality of the implementation of some key error handling features. (As in many standards communities, distinctions sometimes need to be made between conformance with the standard and providing a "high-quality" implementations.)

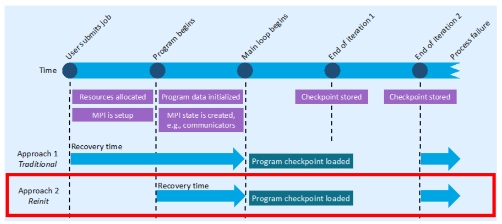


Figure 32: ReInit reduces application recovery time.

We are also continuing to drive forward on a number of fronts that did not make the 4.0 version of the standard, but are still considered important or useful for exascale and the ECP community, particularly in the context of resilience. The ULFM proposal has been updated to reflect those features that have been incorporated into 4.0, as well as the discussions within the Forum. For the complementary Reinit simplified checkpoint/restart proposal (Figure 32), we have carried out a formal verification on the recovery algorithm. This ensures that the protocol, which we plan to propose for a future version of MPI, correctly handles the recovery phase of a failure response, including correct propagation of notifications, absence of deadlocks, and proper termination.

We have provided initial implementations of the features above, and more, within the context of the Open MPI library, to allow users to explore the new capabilities. In addition to sessions and partitioned communications, we have expanded the MPI_T interface implementation to include events, and implemented a number of new events and performance variables, demonstrating their use in the context of ECP miniapplications and tools.

Motivated largely by support for the Partitioned Communications proposal and other situations where high network concurrency is required, a general user-level threading abstraction has been developed to support both the Qthreads and BOLT/Argobots threading libraries within either the Open MPI or MPICH libraries. Work begun last year, in collaboration with the MPICH and Argobots ECP teams (WBS 2.3.1.07 and 2.3.5.05), has now been integrated into the two MPI implementations. An important part of the work within Open MPI involved abstracting the threading layer so that it was not limited to pthreads.

Work on integration with hardware and software environments specifically relevant to the exascale systems continues as well. On the software side, we have implemented support for the ALCF's Cobalt resource management system in Open MPI and PMIx. This work, which was necessitated by a major refactoring of Open MPPI to exclusively use PMIx as its low-level runtime layer, will facilitate support for Aurora, which will use a successor to Cobalt. Combining both software and hardware, we have been working with the SICM ECP project (WBS 2.3.1.16) to integrate support for complex memory hierarchies into Open MPI based on SICM's APIs. This work has now been demonstrated in several contexts within the Open MPI library, including the ability to selectively place objects in either GPU HBM memory or DRAM. We have also provided a design document for library developers to work with different memory pools in the context of Open MPI. Finally, on the hardware side, we have benefit working with HPE (Cray) to ensure that the new partitioned communication capability in MPI 4.0 will support GPUs effectively.

In addition to these activities, we continue to support quality assurance of the Open MPI code base through more and better testing. The Open MPI testing infrastructure, MTT, continues to be improved for flexibility and capability. One of this year's noteworthy efforts was the addition of the bueno application test harness to the testing system. The facilitates incorporating third-party test cases (i.e. based on user applications) into the routine testing process without having to commit them to the Open MPI testing repository.

Next Steps In FY21 and beyond, we plan to continue working across the multiple fronts described above. We will be in a much better position to work with application teams who can benefit from the new capabilities embodied in MPI 4.0, but who have been reluctant to adopt them until they were officially part of the standard. We will likewise continue to identify and improve performance and scalability bottlenecks, and to work with HPE (Cray) and the larger community to ensure that their Slingshot network and GPU support are ready for Frontier and Aurora.

4.1.15 WBS 2.3.1.18 RAJA/Kokkos

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

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

Key Challenges One of the biggest challenges for the Exascale supercomputing era is the proliferation of different computer architectures, and their associated mechanisms to program them. Vendors have an incentive to develop their own models in order to have maximum freedom of exposing special hardware capabilities, and potentially achieve "vendor-lock-in". This poses the problem for applications that they may need to write different variants of their code for different machines - an effort which can be simply not feasible for many of the larger application and library projects.

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

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

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

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

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

Kokkos Support addresses the challenges of establishing, growing and maintaining the user community. First and foremost, it provides explicit means for supporting all DOE ECP applications. A main component of that is funding for local Kokkos experts at the Sandia, Oak Ridge, Argonne, Berkeley and Los Alamos laboratories which can serve as direct contacts for local applications and the users of the leadership computing facilities. Secondly, the project develops and maintains a reusable support infrastructure, which makes supporting more users scalable and cost effective.

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

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

Furthermore, certain areas of common needs between RAJA and Kokkos have emerged. To avoid duplicated efforts, and leverage possible synergies the two teams are developing certain capabilities together. These include for example:

- advanced atomic support with memory order and memory scope exposure.
- common metaprogramming facilities.
- optional integration of Umpire memory pools into Kokkos.
- integration of Kokkos Tools callback mechanisms into RAJA.
- an extension of the RAJA performance test suite to include Kokkos variants.

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

Kokkos is used by many applications in production across the entire spectrum of DOE's super computers. Support for current production platforms is mature and stable. Work on supporting the upcoming Exascale platforms is underway and the primary Kokkos capabilities for AMD GPUs and Intel GPUs are working. Initial application tests were successfully conducted with projects such as EXAALT/LAMMPS, ArborX and Cabana.

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

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

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

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

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

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

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

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

The goal of the Argo project [54] is to augment and optimize existing OS/R components for use in production HPC systems, providing portable, open source, integrated software that improves the performance and scalability of and that offers increased functionality to exascale applications and runtime systems.

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

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

Project activities focus around four products:

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

AML

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

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

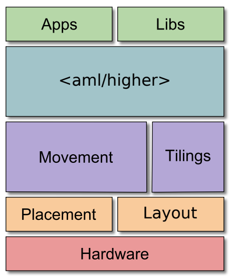
Key Challenges Between non-uniform memory access (NUMA) to regular DRAM, the 3-D stacked high-bandwidth memory, and the memory local to the accelerator devices such as GPUs, the increasing depth of the memory hierarchy presents exascale applications with a critical challenge of how to use the available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

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

Solution Strategy AML provides explicit, application-aware memory management for deep memory systems. It offers a collection of building blocks that are *generic*, *customizable*, and *composable*. Applications can specialize the implementation of each offered abstraction and can mix and match the components as needed. AML can be used to create, for example, a software-managed scratchpad for multilevel DRAM hierarchy such as HBM and DDR. Such a scratchpad provides applications with a memory region with a predictable high performance for critical data structures.

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

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



- Memory *areas*, the location where data lives,
- Applications data *layouts* description,
- *DMA* engines, to move data across areas and layouts,
- *Tiling* schemes, the meta-structures on top of data layouts,
- High level abstractions and helpers (*replicaset*).

Recent Progress This year AML contributions are distributed across new features, infrastructure improvements, and a new collaboration project.

As part of our effort to improve application locality, we merged features on top of the *hwloc backend*. The hwloc library exposes the machine topology and its objects attributes. As a result, AML is able to process the relationship between processing units and memories in terms of latency, bandwidth, and hop distance. Additionally, AML is able to make use of user-provided distances. Hence, it becomes possible to enhance available performance data with benchmarked metrics. From AML user's perspective, a new *area* base block implementation is available. It implements the *preferred* allocation policy on all memories. These memories are ordered according to a performance criterion (e.g., bandwidth). Therefore, data can be allocated as long as some memory is available. On top of that, data will be mapped onto the best available memories according to a performance criterion. Furthermore, we updated and merged the *replicaset* high level abstraction which is now available to users. In a nutshell, this abstraction will create an area per NUMA cluster on the fastest memory of the cluster. Then, the user can use the replicaset primitives to map data replicas and update the latter in these areas. We previously demonstrated the use of this facility to improve the performance of the *XSBench* application (published this year). We are working with ExaSMR developers to offer it as an optional feature in the latest version of the benchmark.

Finally, the last new capability we introduced in the library is a submodule library called *excit*. Excit implements general-purpose iterators, and we are using it to provide extensive iteration interfaces to the core building blocks of AML, as well as custom topology object iterators. For example, we want to be able to iterate memories of a certain type or with specific performance abilities.

As part of our infrastructure improvements, we purchased a new machine with heterogeneous hardware that we are using in our CI pipeline to test new vendor backends. We are also taking advantage of the ECP CI infrastructure as part of our development process. The development of the library is validated by our own CI before being merged into a staging branch. The latter is then mirrored to ECP facilities to be further tested by an ECP CI pipeline, before being merged into our master branch. The goal is to ensure that our master branch can always be stable on production systems. Our ECP CI pipeline runs on Theta at ALCF and we are in the process of setting it up at OLCF.

Next Steps We are looking to build on top of our success with XSBench and use it as a showcase of some of the AML capabilities. Our next goal is to provide a feature inspired by OpenMP 5 *custom mappers*. An OpenMP mapper makes it possible to describe which parts of a structure should be mapped onto accelerator devices. As it is a recent addition to the standard, to the best of our knowledge no compiler implements it yet. Furthermore, this feature is limited in its ability to filter or reorganize memory during the mapping process. We are using our *tiling* abstraction to build a more flexible interface for such a feature, based on the needs of the OpenMC application. It will be combined with our *DMA* abstraction to manage memory transfers and might be also provided through a more straightforward higher-level API.

Finally, we are aiming at a broad compatibility of the library with accelerator devices. Therefore, we are about to implement the base library abstractions on top of the *OpenCL* backend.

UMap

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

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

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

Solution Strategy We prioritize four design choices for UMap based on surveying realistic use cases. First, we choose to implement UMap as a user-level library so that it can maintain compatibility with the fast-moving Linux kernel without the need to track and modify for frequent kernel updates. Also, we employ the recent userfaultfd mechanism, rather than the signal handling + callback function approach to reduce overhead and performance variance in multi-threaded applications. Third, we target an adaptive solution that sustains performance even at high concurrency for data-intensive applications, which often employ a large number of threads for hiding data access latency. Our design pays particular consideration on load imbalance among service threads to improve the utilization of shared resources even when data accesses to pages are skewed. UMap dynamically balances workloads among all service threads to eliminate bottleneck on serving hot pages. Finally, for flexible and portable tuning on different computing systems, UMap provides both API and environmental controls to enable configurable page sizes, eviction strategy, application- specific prefetching, and detailed diagnosis information to the programmer. The UMap software architecture is shown in Figure 33.

Recent Progress In recent months, we have released UMap v2.1.0 and updated the UMap Spack package with major enhancements to improve load balancing and incorporate features for additional configurability. The new features include a SparseStore handler to optimize access to sparse, randomly accessed persistent data structures. The MetaAllocator Metall developed under the ECP SICM project has been integrated into UMap's SparseStore, enabling persistent heaps.

UMap now supports a new network-based handler using the ECP Mochi data service to fetch and manage memory pages from remote nodes over the network. UMap also has a new capability to provide region-centric page access profiling from the collaboration with the ECP Caliper project.

We have incorporated UMap into new applications, including the Livermore Metagenomics Toolkit (LMAT), which is being used for COVID research. We designed custom prefetching and eviction policies for LMAT. The new solution outperforms the system solution in runtime by $5-15\%$ on realistic metagenomic queries and supports high parallelism that cannot be supported efficiently by the system solution.

A paper on the UMap network handler has been accepted and presented at the IEEE SBAC-PAD 2020 conference. An SC20 poster on the UMap SparseStore handler is to appear.

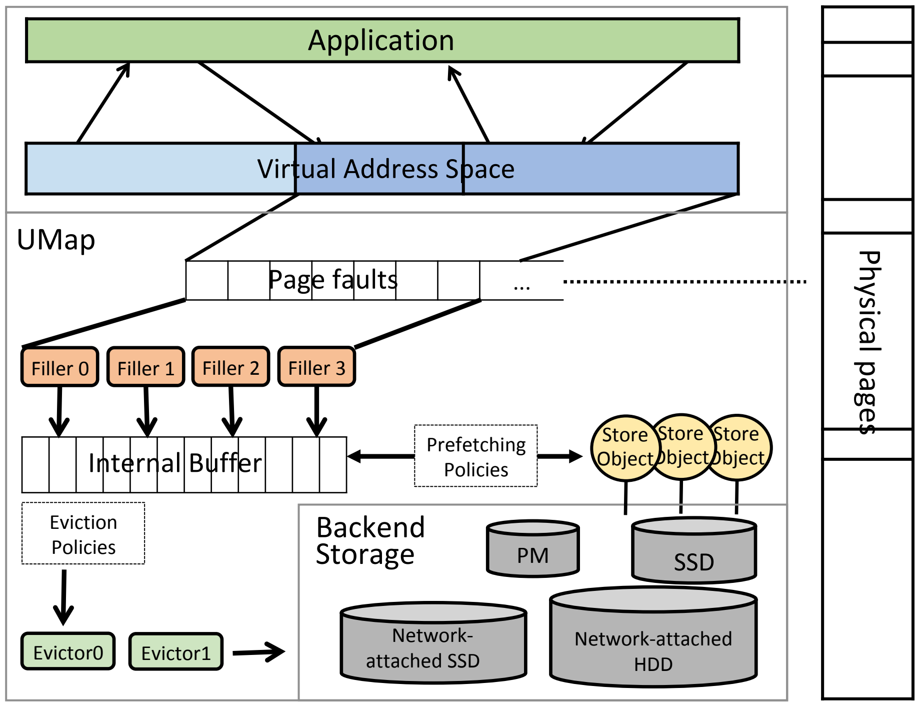


Figure 33: UMap Handler architecture

Next Steps In the coming year, we plan to continue outreach to application teams within ECP and in the science/data community. This includes collaboration with Caliper, Mochi, Exagraph, the LMAT team, and with LLNL users of materials and EOS tables. The Exagraph collaboration will map persistent data such as binary format graphs and intermediate data structures through UMap using the SparseStore and Metall. To adapt to dynamic changes in memory resources, an adaptive buffer management will be added to monitor and react at runtime. To support the shared table use case, we will implement a multi-process version of UMap to be used by multiple MPI processes on a node to share read-only tables stored in persistent memory.

PowerStack

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

The ECP PowerStack sub-area in Argo explores hierarchical interfaces for power management at three specific levels: batch job schedulers, job-level runtime systems, and node-level managers. Each level will provide options for adaptive management depending on requirements of the supercomputing site under consideration. Site-specific requirements such as cluster-level power bounds, user fairness, or job priorities will be translated as inputs to the job scheduler. The job scheduler will choose power-aware scheduling plugins to ensure compliance, with the primary responsibility being management of allocations across multiple users and diverse workloads. Such allocations (physical nodes and job-level power bounds) will serve as inputs to a fine-grained, job-level runtime system to manage specific application ranks, in-turn relying on vendor-agnostic node-level measurement and control mechanisms. The figure below presents an overview of the envisioned PowerStack, which takes a holistic approach to power management. Additionally, power management support

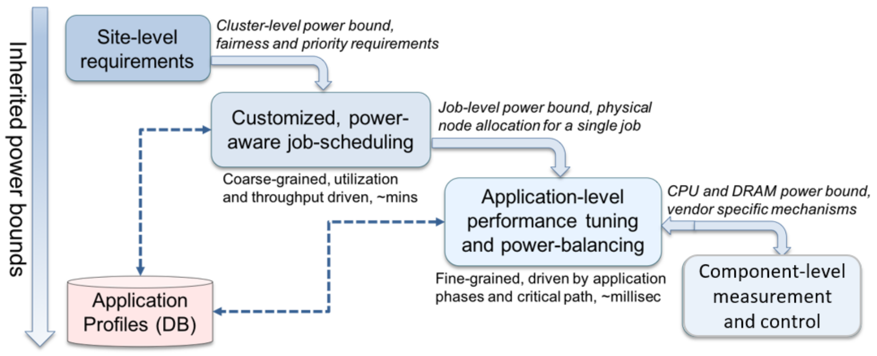

Inherited power bounds
Site-level requirements
Cluster-level power bound, fairness and priority requirements
Customized, power-aware job-scheduling
Job-level power bound, physical node allocation for a single job
Coarse-grained, utilization and throughput driven, ~mins
Application-level performance tuning and power-balancing
CPU and DRAM power bound, vendor specific mechanisms
Fine-grained, driven by application phases and critical path, ~millisec
Component-level measurement and control
Application Profiles (DB)

Figure 34: Envisioned PowerStack

for science workflows (such as MuMMI Cancer workflow, E3SM climate models, etc), in-situ visualization, and workflow management infrastructures (e.g. Kokkos and Caliper) is being developed. Interfaces with ATDM projects such as LLNL's Flux are also being developed. Furthermore, solutions for co-scheduling challenges for extremely heterogeneous architectures are being designed as a part of a university subcontract to University of Arizona.

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

Key Challenges Power management in software is challenging due to the dynamic phase behavior of applications, processor manufacturing variability, and the increasing heterogeneity of node-level components. While several scattered research efforts exist, a majority of these efforts are site-specific, require substantial programmer effort, and often result in suboptimal application performance and system throughput. Additionally, these approaches are not production-ready and are not designed to cooperate in an integrated manner. A holistic, generalizable and extensible approach is still missing in the HPC community, and a goal for the ECP Argo PowerSteering project is to provide a solution for this technology gap.

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

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

Recent Progress We achieved three milestones through FY20 in September 2020. The first was to release a production-ready version of Variorum, a vendor neutral power monitoring and control library, with support for Intel, IBM and NVIDIA platforms. Variorum currently supports IBM Witherspoon architecture with Power9, NVIDIA Volta GPUs, and six Intel microarchitectures (Sandy Bridge, Ivy Bridge, Haswell, Broadwell, Kaby Lake, Skylake) in a vendor-neutral manner. Our second milestone this year was the initial deployment and testing of a production-ready PowerStack. Here, we first developed interfaces between Intel GEOPM and Variorum to allow for vendor-neutral access across the platforms that Variorum supports. Then, we tested and included GEOPM v1.1 as a part of the TOSS (Tri-Lab Operating System) release, which included significant testing and integration. This now allows users on the Tri-Lab systems with the underlying platforms (currently Intel) to utilize GEOPM. Additionally, we implemented SLURM SPANK plugins through msr-safe to allow for resource management level control. We also tested our Flux variation-aware plugin, which was developed last year as part of ECP Argo for the second milestone. For our third milestone, we carried out power and performance analysis of the MuMMI Cancer Workflow, explored power management of co-scheduled applications by studying NAS benchmarks and E3SM configurations, and delivered a Kokkos-tool extension to Variorum. We began the process of understanding how we can write a Caliper service for Variorum as well. In parallel, as part of ASC L2 for the Flux effort, Flux and Variorum integration was also carried out for upcoming supercomputers for both power monitoring and capping (outside of the scope of this ECP Argo PowerSteering project, but expected to help with integration efforts at exascale). Additionally, LLNL is working on a multi-vendor CRADA involving Intel, HPE, ARM and IBM – industry partners that are helping us drive vendor-neutral solutions to power management; and are actively engaged in the community effort for PowerStack homogenization. We are also working in collaboration with PowerAPI team for the same. We established the PowerStack community charter in June 2018, involving collaborators across multiple vendors (Intel, IBM, ARM, HPE, AMD, NVIDIA, Cray), academic institutions (TU Munich, Univ. Tokyo, Univ. Bologna, Univ. Arizona), and national laboratories (Argonne National Lab).The goal for this team is to design a holistic, flexible and extensible concept of a software stack ecosystem for power management. Over the past 2.5 years, this group is looking at engineering and research challenges, along with RFP/procurement designs through active vendor interaction and involvement. We held a 3-day seminar in Nov 2019 and again in June 2020 virtually this year, the details of which can be found here: `https://hpcpowerstack.github.io/powerstack-nov19.html`.

Next Steps We will continue our research and development work as planned toward the FY21 milestones. More specifically, we will continue development for variorum library to allow support for ARM, AMD and other architectures. We will continue to extend Intel GEOPM's new codebase, continue development of scheduler components such as Flux and SLURM, work on GPU power capping research, and enable user-space access to power management on diverse architectures. We will expand our collaborations for science workflows, such as MuMMI and E2SM, including support for Caliper and Kokkos power management. We will also continue to further explore co-scheduling challenges in power management (University of Arizona) and multi-tenancy issues in power management on heterogenous architectures, and lead the efforts on multi-vendor CRADA.

NRM

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

NRM interacts with both global resource management services (e.g., job scheduler) and with application components and runtime services running on the node. It acts as a control infrastructure to enable custom resource management policies at the node level. Applications can influence these mechanisms, both directly (through explicit API calls used, e.g., to request additional resources on the node) and indirectly (by having their run-time behavior monitored by NRM).

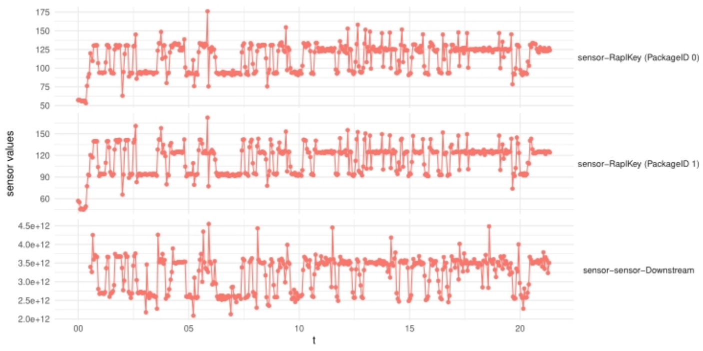
Key Challenges Many ECP applications have a complex runtime structure, ranging from in situ data analysis, through an ensemble of largely independent individual subjobs, to arbitrarily complex workflow structures. At the same time, HPC hardware complexity increases as well, from deeper memory hierarchies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

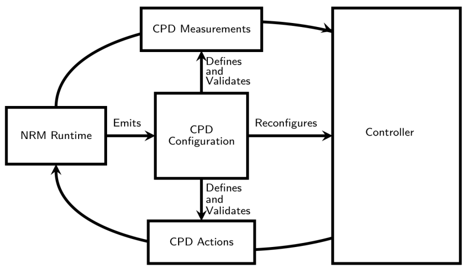
Even in the common case of each compute node being allocated exclusively to a single job, managing available node resources can be a challenge. If a compute node is shared among multiple job components (a likely scenario considering the reduced cost of data transfers), these components—if allowed to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

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

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

In addition to those capabilities, NRM's configuration format allows for the configuration of actuators and active polling sensors based on arbitrary executables. The figure on the right shows a reading of NRM's sensor outputs for two RAPL sensors along with CPU counter instrumentation. In order to provide these capabilities, the NRM daemon manages the launching, management, and stopping of parallel workloads through a unified interface. Resources can be dynamically reconfigured at run time; these interfaces are provided for use from applications and from global services.





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

Recent Progress The NRM daemon has undergone major improvements. In order to improve the software quality and overall reliability of NRM, the core business logic of the NRM daemon has been externalized in a shared library, written in the statically typed language Haskell. This NRM version is pending an upcoming release. Our custom CI pipeline also has been improved, with the addition of unit tests and enrichment of existing functional tests. We are still in the process of leveraging ECP testing infrastructure.

NRM has recently added support for multi-armed bandit based control policies. In order to make this work as robust as possible, it has been compartmentalized into a software library (*hbandit*). *hbandit* implements statically typed, independently tested versions of the relevant machine learning algorithms.

NRM now offers a Python support library for upstream control. This library is intended for workload-specific control algorithm experiments.

We continue supporting the `libnrm` C library that can be linked to applications in order to provide reports on application progress to NRM. We have instrumentation for EXAALT, QMCPACK, ExaSMR, AMG, Stream, and CANDLE. This capability gives us insight into the effect of our resource management policies on the run-time behavior of user codes.

Next Steps We are working on a report to showcase the use of NRM's current autonomic resource management strategies in the context of workload energy minimization, through the use of RAPL control and application feedback.

We are also working on expanding the set of ECP applications that are instrumented to report their progress to NRM, as well as validation of the infrastructure on multiple ECP platforms.

We are planning to expand the list of resources managed by NRM by adding support for other vendor-specific mechanisms, as well as better integration with other ECP system software (job schedulers, application tracing, and power interfaces).

4.2 WBS 2.3.2 DEVELOPMENT TOOLS

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

4.2.1 Scope and Requirements

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

4.2.2 Assumptions and Feasibility

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

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

4.2.3 Objectives

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

4.2.4 Plan

Multiple projects will be supported under the tools effort. To ensure relevance to DOE missions, most of these efforts shall be DOE laboratory led and leverage and collaborate with existing activities within the broader HPC community. Initial efforts will focus on identifying the core capabilities needed by the selected ECP applications, components of the software stack, expected hardware features, and the selected industry activities from within the Hardware and Integration focus area. The supported projects will target and implement early versions of their software on both CORAL and APEX systems, with an ultimate target of production-ready deployment on the Exascale systems. Throughout this effort the applications teams and other elements of the software stack will evaluate and provide feedback on their functionality, performance, and robustness. These goals will be evaluated yearly (or more often as needed) based on milestones as well as joint milestone activities shared across the associated software stack activities by AD and HI focus areas.

4.2.5 Risk and Mitigation Strategies

A primary risk exists in terms of adoption of the various tools by the broader community, including support by system vendors. Past experience has shown that a combination of laboratory-supported open source software and vendor-optimized solutions built around standard APIs that encourage innovation across multiple platforms is a viable approach, and this will be undertaken. We will track this risk primarily via the risk register.

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

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

4.2.6 Future Trends

Future architectures are becoming more heterogeneous and complex [55]. As such, the role of languages, compilers, runtime systems, and performance and debugging tools will becoming increasingly important for productivity and performance portability. In particular, our ECP strategy focuses on improving the open source LLVM compiler and runtime ecosystem; LLVM has gained considerable traction in the vendor software community, and it is the core of many existing heterogeneous compiler systems from NVIDIA, AMD, Intel, ARM, IBM, and others. We foresee that this trend will continue, which is why we have organized the Development Tools technical area around LLVM-oriented projects. We expect for many of our contributions to LLVM to address these trends for the entire community and will persist long after ECP ends. For example, our contributions for directive-based features for heterogeneous computing (e.g., OpenMP, OpenACC) will not only provide direct capabilities to ECP applications, but it will also impact the redesign and optimization of the LLVM infrastructure to support heterogeneous computing. In a second example, Flang (open source Fortran compiler for LLVM; [the second version is also known as F18]) will become increasingly important to the worldwide Fortran application base, as vendors find it easier to maintain and deploy to their own Fortran frontend (based on Flang). Furthermore, as Flang become increasingly robust, researchers and vendors developing new architectures will have immediate access to Flang, making initial Fortran support straightforward in ways similar to what we are seeing in Clang as the community C/C++ frontend.

4.2.7 *WBS 2.3.2.01 Development Tools Software Development Kits*

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

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

Solution Strategy The primary mechanism for mitigating risk in the SDK is the *Readiness Survey*. This survey is designed to assess the current status of each product in the SDK in six key areas: software availability, documentation, testing, Spack build support, SDK integration, and path forward technology utilization. By periodically assessing the progress of the individual L4 products in the SDK, we will use the survey to identify and resolve current hardware architecture dependencies, plan for future architecture changes, and increase adoption of the Continuous Integration (CI) testing workflow to reduce this risk.

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

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

Recent Progress The Readiness Survey was re-issued to each L4 product in July 2020. All projects had nominal changes except that HPCToolkit and TAU have added support for both Intel and AMD GPUs. With these changes, four of the six L4 projects support NVIDIA GPUs, 3/6 support AMD, 2/6 support Intel, and 2/6 have support for all three GPU platforms.

The first step in assessing buildability of the L4 products was carried out in Q2 of FY20. All six projects were built on three platforms (Power9, x86, and Aarch64) using gcc and clang. Initially, only four successfully built with gcc on all platforms, and only one built with clang on any platform. The issues were reported and fixed, and now all six products build with gcc successfully on all three pre-exascale test platforms. The clang builds encountered a large number of issues in both the SDK products and their dependencies. The effort required to fix these issues was larger than the time allocated for the task, so they have been moved to Q1 FY21 as part of a larger procedure to assess LLVM/clang compatibility.

Support for automated testing remains a challenge area that all of the projects are aware of and plan to dedicate time to in FY21 and beyond. In Q2 of FY20, initial assessment of testing capabilities was carried out for two of the six products, Dyninst and TAU, on pre-exascale systems. With this work, both products now have working test suites that can be employed through scriptable executions. This represents an essential component of software sustainability to demonstrate and track correctness in the presence of code changes for these products.

Continuous Integration (CI) testing remains a still-larger challenge for the SDK. This is due in part to some products not having scriptable testing capabilities and also in part to more general challenges of using CI at the facilities through OSTI. With substantial help from Don Magrack at NMC and the CI team at ALCF, the first CI run was successfully carried out using Dyninst using Theta at the Argonne Leadership Compute Facilities in Q4 2020. We note that automated CI testing through OSTI remains as future work until the federated runners are established at the national labs.

Next Steps Additional testing using multiple compilers- including some variant of LLVM currently in use by the Compilers and Debuggers SDK- on at least one current DOE facility machine, and preferably one early access system, is our top priority for FY21. Results from these tests will continue to be fed back into the L4 products to further guide development of spack packages, bug/issue-reporting workflows, and integration into the greater ECP software ecosystem. Any discovered issues with Spack, compilers, or libraries will be directly reported back to their respective development teams or L3 representative.

Our KPP3 goals are tightly associated with integrating each L4 product into E4S, and a large part of these goals is getting more products using CI testing. As such, increasing the number of L4 products in the SDK with CI testing adoption is our second goal for FY21. The first CI workflow integration carried out in Q4 2020 established the basic procedure for getting the other products set up. Arguably, establishing this workflow is the largest contribution the Tools SDK will bring to the overall ECP software ecosystem. Having automated testing in place across heterogeneous build environments and target architectures is a fundamental challenge to creating reliable, sustainable software- making this work a critical path to attaining the ECP goals of large-scale software sustainability. We also anticipate that this may be the introduction of formal software testing for some of the L4 products. The heterogeneous nature of the testing available in the Tools SDK L4 products will serve as a focused testbed for constructing implementation guidelines for the CI workflow which can then be applied across the SDK efforts and into the greater ECP software ecosystem. Importantly, these lessons can also be carried on by the individual project teams to help maintain their software beyond the ECP timeline.

4.2.8 WBS 2.3.2.06 *Exa-PAPI*

Overview The Exa-PAPI project is developing a new C++ Performance API (PAPI++) software package from the ground up that offers a standard interface and methodology for using low-level performance counters in CPUs, GPUs, on/off-chip memory, interconnects, and the I/O system, including energy/power management. PAPI++ is building upon classic-PAPI functionality and strengthening its path to exascale with a more efficient and flexible software design, one that takes advantage of C++'s object-oriented nature but preserves the low-overhead monitoring of performance counters and adds a vast testing suite.

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

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

Solution Strategy The Exa-PAPI team is preparing PAPI support to stand up to the challenges posed by exascale systems by

1. widening its applicability and providing robust support for exascale hardware resources;
2. supporting finer-grain measurement and control of power, thus offering software developers a basic building block for dynamic application optimization under power constraints;
3. extending PAPI to support software-defined events; and
4. applying semantic analysis to hardware counters so that the application developer can better make sense of the ever-growing list of raw hardware performance events that can be measured during execution.

In summary, the team will be channeling the monitoring capabilities of hardware counters, power usage, software-defined events into a robust PAPI++ software package. PAPI++ is meant to be PAPI's replacement—with a more flexible and sustainable software design.

Recent Progress The Exa-PAPI team shipped the PAPI 6.0.0 release on March 4, 2020. This release includes a new API for Software Defined Events (SDEs), a major revision of the high-level API, and several new components, including ROCm and ROCm_SMI for AMD GPUs, powercap_ppc and sensors_ppc for IBM Power9 and later, the SDE component to expose software-defined events through the standard PAPI interface, and the IO component that exposes I/O statistics exported by the Linux kernel.

Furthermore, PAPI 6.0.0 ships CAT, a new Counter Analysis Toolkit that assists with native performance counter disambiguation through micro-benchmarks. Performance in different CPU architectures can be monitored by reading the occurrences of various hardware events. However, from architecture to architecture, it is not clear which native hardware events are indexed by which event names, making it difficult for the performance analyst to understand how to measure specific events. To alleviate this difficulty, CAT aims to measure the occurrences of events through a series of benchmarks, classify particular events of interest via data analysis of the patterns of event occurrences, and allow its users to discover the high-level meaning of native events. The examples in Figure 35 show the results from the CAT floating-point tests used to verify single- and double-precision FLOP counters on three different CPU architectures (Intel Broadwell, Intel Skylake, IBM POWER9 CPU). Our paper [56] discussed how to monitor bandwidth and arithmetic intensity with PAPI and CAT.

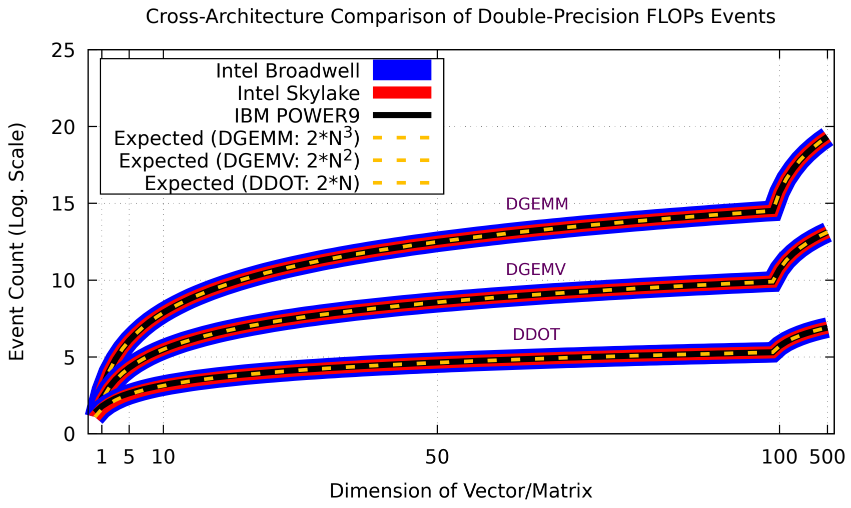


Figure 35: FLOPs validation on Broadwell, Skylake, and POWER9.

For the refactoring PAPI to PAPI++ effort, the Exa-PAPI team circulated a survey to the ECP applications (AD) and software technology (ST) teams to assess their needs and requirements for hardware and software performance counter functionality. A white paper [57] has been released, describing our survey findings and articulating research priorities for the development of the new PAPI++ software package. In July 2020, a second white paper [58] has been released, describing our roadmap for refactoring PAPI to PAPI++. The objective of these documents is to describe the current design of the classic PAPI framework in conjunction with its limitations and sustainability challenges, as well as identify and articulate opportunities and possible implementations and C++ features for PAPI++ to provide support for the simultaneous measurement of data from multiple counter domains.

Next Steps Our next efforts will focus on:

1. **Production-ready CMake build system for PAPI++:** Implementation of a production-ready CMake build system for PAPI++ . We will build on the prototype CMake implementation from classic PAPI (produced in STTO09-103), and use the "PAPI Red Team" input and feedback for implementing a CMake build system for the new PAPI++ framework and its component.

2. **Transition from PAPI's C framework to the new PAPI++ framework implemented in C++**: Implementation of the modular structure of the new PAPI++ framework that includes a new C++ API in addition to the traditional C and Fortran APIs to preserve backward-compatibility. Focus will be on the implementation of the PAPI++ core and modular framework that is architecture-neutral. We will build on the design and prototype implementation from STTO09-105. A beta release for early users will be planned as completion criteria. It will allow us to follow an iterative and incremental software development process early in the design cycle before adding all of the 30+ PAPI components to the new PAPI++ framework.

3. **Development of a C++ API for PAPI's stand-alone SDE library**: In addition to the already implemented and released C and FORTRAN APIs for the PAPI software-defined event (SDE) functionality (release with STTO09-104), we will develop a C++ API for the stand-alone SDE library while maintaining its traditional APIs. The goal is to achieve consistency by providing all three APIs (C++ , C, FORTRAN) across all PAPI++ modules. This effort involves transitioning PAPI SDE to the new PAPI++ package.

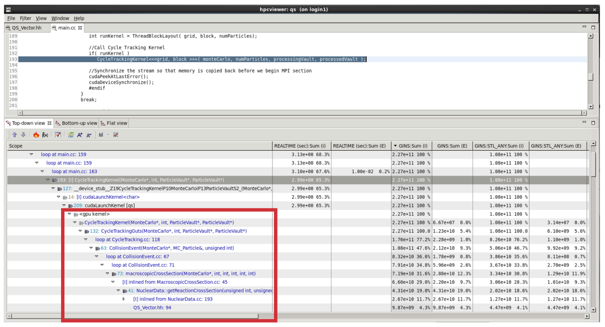
*4.2.9 WBS 2.3.2.08 **HPCToolkit***

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

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

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

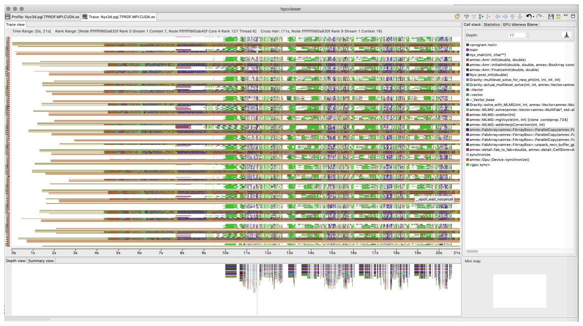
Solution Strategy Development of HPCToolkit as part of ECP is focused on preparing it for production use at exascale by enhancing it in several ways. First, the team is adding new capabilities to measure and analyze interactions between software and key hardware subsystems in extreme-scale platforms,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



```
hpcviewer: qs (on login1)
File Filter View Window Help
QS_Vector.hh   main.cc

                    int runKernel = ThreadBlockLayout( grid, block, numParticles);

                    //Call Cycle Tracking Kernel
                    if( runKernel )
                       CycleTrackingKernel<<<grid, block >>>( monteCarlo, numParticles, processingVault, processedVault );

                    //Synchronize the stream so that memory is copied back before we begin MPI section
                    cudaPeekAtLastError();
                    cudaDeviceSynchronize();
                    #endif
                 }
                 break;
```

```
Top-down view   Bottom-up view   Flat view

Scope                                                                     REALTIME (sec):Sum (I)  REALTIME (sec):Sum (E)  GINS:Sum (I)     GINS:Sum (E)     GINS:STL_ANY:Sum (I)  GINS:STL_ANY:Sum (E)
  loop at main.cc: 159                                                    3.13e+00 68.3%                                  2.27e+11 100 %                    1.08e+11 100 %
    loop at main.cc: 159                                                  3.13e+00 68.3%                                  2.27e+11 100 %                    1.08e+11 100 %
      loop at main.cc: 163                                                3.10e+00 67.6%          1.00e-02  0.2%          2.27e+11 100 %                    1.08e+11 100 %
        193: [I] CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVault*)  2.99e+00 65.3%                    2.27e+11 100 %                    1.08e+11 100 %
          127: __device_stub__Z19CycleTrackingKernelP10MonteCarloiP13ParticleVaultS2_(MonteCarlo*,  2.99e+00 65.3%        2.27e+11 100 %                    1.08e+11 100 %
            14: [I] cudaLaunchKernel<char>                                2.99e+00 65.3%                                  2.27e+11 100 %                    1.08e+11 100 %
              209: cudaLaunchKernel [qs]                                  2.99e+00 65.3%                                  2.27e+11 100 %                    1.08e+11 100 %
                <gpu kernel>                                                                                              2.27e+11 100 %                    1.08e+11 100 %
                  CycleTrackingKernel(MonteCarlo*, int, ParticleVault*, ParticleVault*)                                   2.27e+11 100 %   6.67e+07  0.0%   1.08e+11 100 %        3.14e+07  0.0%
                    132: CycleTrackingGuts(MonteCarlo*, int, ParticleVault*, ParticleVault*)                              2.27e+11 100.0   1.23e+10  5.4%   1.08e+11 100.0        6.10e+09  5.6%
                      loop at CycleTracking.cc: 118                                                                       1.76e+11 77.2%   2.28e+09  1.0%   8.26e+10 76.2%        1.10e+09  1.0%
                        63: CollisionEvent(MonteCarlo*, MC_Particle&, unsigned int)                                       1.08e+11 47.6%   2.12e+10  9.3%   5.06e+10 46.7%        9.92e+09  9.2%
                          loop at CollisionEvent.cc: 67                                                                   8.32e+10 36.6%   1.78e+09  0.8%   3.86e+10 35.6%        8.11e+08  0.7%
                            loop at CollisionEvent.cc: 71                                                                 7.91e+10 34.8%   5.96e+09  2.6%   3.67e+10 33.8%        2.70e+09  2.5%
                              73: macroscopicCrossSection(MonteCarlo*, int, int, int, int, int)                           7.19e+10 31.6%   2.80e+10 12.3%   3.34e+10 30.8%        1.29e+10 11.9%
                                [I] inlined from MacroscopicCrossSection.cc: 45                                           6.60e+10 29.0%   2.20e+10  9.7%   3.06e+10 28.3%        1.01e+10  9.3%
                                  41: NuclearData::getReactionCrossSection(unsigned int, unsigned                         4.31e+10 19.0%   4.31e+10 19.0%   2.02e+10 18.6%        2.02e+10 18.6%
                                    [I] inlined from NuclearData.cc: 193                                                  2.67e+10 11.7%   2.67e+10 11.7%   1.27e+10 11.7%        1.27e+10 11.7%
                                    QS_Vector.hh: 94                                                                      9.87e+09  4.3%   9.87e+09  4.3%   4.47e+09  4.1%        4.47e+09  4.1%
```

(a)



(b)

Figure 36: (a) HPCToolkit's `hpcviewer` showing a detailed attribution of GPU performance metrics in a profile of Quicksilver. (b) HPCToolkit's `hpctraceviewer` showing CPU and GPU trace lines for Nyx.

Recent Progress

- The project team developed a unified GPU monitoring substrate. Upon this substrate, they developed support for collecting summary metrics for kernel launches, memory copies, and synchronizations on AMD, Intel, and NVIDIA GPUs.
- The project team enhanced support for fine-grained measurement of GPU computations using PC sampling on NVIDIA GPUs and binary instrumentation on Intel GPUs. HPCToolkit uses instruction-level measurements to reconstruct approximate calling context trees to help developers understand performance of complex GPU kernels. Figure 36(a) shows a calling context for Quicksilver, an ECP proxy application for LLNL's Mercury next-generation transport code; the highlighted region shows an approximate reconstruction of a GPU calling context tree.
- The project team enhanced HPCToolkit for collecting and visualizing traces of both CPU and GPU activity. Figure 36(b) shows a trace of a GPU-accelerated execution of a recent snapshot of Nyx—an adaptive mesh, compressible cosmological hydrodynamics simulation code being developed as part of the ECP ExaSky project. Each GPU kernel is related to the CPU calling context in which it was launched, which helps developers understand the performance of complex applications.
- The project team developed a novel approach that utilizes both shared and distributed memory parallelism to analyze and aggregate sparse data representations of performance measurements from every rank, thread and GPU stream in a program execution.
- The project team improved the reliability of HPCToolkit's hpcrun for monitoring complex dynamic library loading and unloading and interacting with the application when other software tools are present.

Next Steps

- Integrate GPU measurement and analysis capabilities into HPCToolkit's trunk for release.
- Continue to improve solutions for fine-grained measurement of computations on Intel GPUs and explore new solutions for fine-grained measurement on AMD GPUs.
- Continue to refine the reliability and performance of monitoring operations on dynamic libraries.
- Work with the open source community to upstream GPU measurement support developed by the project team into the community version of the `libomptarget` offloading library.
- Work with DOE and platform vendors to evaluate and refine software interfaces for measuring GPU performance.

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

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

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

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

- Improve the core-LLVM compiler ecosystem;
- Design and implement the OpenACC heterogeneous programming model for C/C++ in Clang/LLVM (Clacc);
- Design and implement the OpenACC heterogeneous programming model for Fortran in Flang/LLVM (Flacc);
- Use performance modeling and optimization to enable code transformation and performance portability;
- Refine autotuning for OpenMP and OpenACC programming models in order to directly target challenges with heterogeneous architectures;
- Improve performance measurement and analysis tools (TAU) for the target exascale architectures and apply it to applications to improve performance;
- Develop and implement portable software abstractions (Papyrus) for managing persistent memory;
- Aggressively engage applications, SDK, vendor, and software teams to demonstrate and deploy; and,
- in collaboration with SOLLVE and Flang, develop a DOE ECP fork of LLVM that will be the clearinghouse for ECP modifications of LLVM (see `https://github.com/llvm-doe-org`).

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

Recent Progress: Our recent work has focused on six topics:

1. OpenACC and Clacc [59]. Develop production-quality, standard-conforming OpenACC compiler and runtime support as an extension of Clang/LLVM. See §4.2.12.
2. OpenACC and Flacc. Develop production-quality, standard-conforming OpenACC compiler and runtime support as an extension of Flang/LLVM.
3. Performance analysis with Tau by adding additional functionality for new architectures. Improve a widely-used performance analysis framework by adding functionality for new architectures and software systems. See §4.2.17.
4. Improving LLVM. In collaboration with numerous other ECP projects, PROTEAS is contributing improvements to the LLVM compiler infrastructure. These improvements include simple bugfixes to the existing infrastructure, monitoring Flang progress, developing Clacc (see §4.2.12), developing Flacc (See §4.2.14), and developing a DOE ECP fork of LLVM for our work.
5. Papyrus [60, 61] for portability across NVM architectures. Develop a portable interface to NVM architectures to provide massive, persistent data structures as required by many applications. See §4.2.18.
6. Outreach and collaboration with ECP applications teams. We have interacted with over a dozen applications teams to help prepare their applications for ECP. See §4.2.12, §4.2.18, and §4.2.17.

Next Steps: Our next efforts are:

1. Clacc. Continue developing OpenACC support by lowering OpenACC directives to use the existing LLVM OpenMP infrastructure.
2. Flacc. Continue developing OpenACC support by finishing the development of the OpenACC dialect for MLIR and beginning to develop the runtime system on the existing LLVM OpenMP infrastructure.
3. Papyrus. Improve support for versioning and other performance improvements.
4. Tau. Improve performance instrumentation for deep memory hierarchies in Tau, focusing primarily on various GPUs and emerging NVM.
5. ECP LLVM fork. Finish consolidation of ECP activities into the ECP LLVM fork, and start basic support for continuous integration.

4.2.11 WBS 2.3.2.10 ***PROTEAS-TUNE: LLVM***

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

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

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

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

There are two remaining challenges that we are addressing: The first is that deploying autotuning relying on source-to-source transformations is difficult because maintaining these separate source kernel versions is practically difficult. The second is that, as a general matter, performance improvements can be obtained by specializing code and runtime as opposed to limiting ourselves to ahead-of-time code generation.

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

- Enhancements to LLVM's inter-procedural analysis, and an improved representation of parallelism constructs, to allow LLVM to propagate information across boundaries otherwise imposed by parallelism constructs, and to allow LLVM to transform the parallelism constructs themselves.

- Enhancements to LLVM's loop-optimization infrastructure to allow the direction of a sequence of loop transformations to loop nests, exposing these features to users through Clang pragmas (in addition to being available at an API level to tools such as autotuners), enabling those transformations to execute as specified, and otherwise enhancing the loop-optimization infrastructure.

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

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

Recent Progress For parallelism, we have implemented several new features in upstream LLVM including an OpenMP-aware optimization pass that performs various optimizations specific to OpenMP code on the host and device (=GPU) [62]. It is run by default with medium and high optimizations enabled ("-O2" and "-O3"). In addition to transformations it will provide user feedback in form of optimization remarks ("-Rpass=openmp-opt")

Extension to the Attributor inter-procedural optimization framework that transparently applies transformations across the boundary between sequential and parallel code. This upstream work will reduce the overhead parallelism introduces due to missed classical optimizations [63].

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

For loop optimizations, we have implemented several new features in LLVM and Clang, including the OpenMP 5.1 "tile" directive and clang pragma syntax for exploration of future transformations not yet available through the OpenMP standard. Most of these enhancements are in papers ([64, 65] and in several forums directly to the LLVM community.

In a more forward looking approach we prototyped a loop-hierarchical IR for LLVM which we also present and discuss in various LLVM community forums.

To facilitate autotuning (ref. Section 4.2.15), we implemented a loop nest information extraction tool for LLVM-IR [66].

We have developed a prototype C++ compiler, based on Clang, supporting an extension that enables programmers to embed (domain specific) languages inside their "C-like" programs [67]. That is, we allow a new type of Clang plugin to bridge the gap between classical code, e.g., C or C++, and code written in a different language, e.g., a quantum or tensor domain specific language (DSL). The latter two examples have been successfully prototyped as well.

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

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

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

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

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

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

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

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

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

Despite the importance of OpenACC, the only non-academic open-source OpenACC compiler cited by the OpenACC website is GCC [68]. However, GCC has lagged behind commercial compilers, such as PGI's, in providing production-quality support for the latest OpenACC specifications [69]. Moreover, GCC is known within the compiler community to be challenging to extend and, especially within the DOE, is losing favor to Clang and LLVM for new compiler research and development efforts.

Clacc [59] is a major component of the PROTEAS-TUNE project. Overall, the goal is to build on Clang and LLVM to develop an open-source, production-quality OpenACC compiler ecosystem that is easily extensible and that utilizes the latest research in compiler technology. Such an ecosystem is critical to the successful acceleration of ECP applications using modern HPC hardware. The PROTEAS-TUNE objectives for Clacc are:

1. Develop production-quality, standard-conforming OpenACC compiler and runtime support as an extension of Clang and LLVM. Two compilation modes are being developed: (a) traditional mode, which produces a binary, and (b) source-to-source mode, which produces OpenMP source.

2. As part of the design, leverage the Clang ecosystem to enable the future construction of source-level OpenACC tools, such as pretty printers, analyzers, lint tools, debugger extensions, and editor extensions.

3. Throughout development, actively contribute improvements to the OpenACC specification, and actively contribute mutually beneficial improvements to the upstream Clang and LLVM infrastructure.

4. As the work matures, contribute OpenACC support itself to upstream Clang and LLVM so that it can be used by the broader HPC and parallel programming communities.

Key Challenges

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

2. **Source-to-Source:** Source-to-source translation to OpenMP significantly reduces the effort to implement OpenACC and offers additional capabilities, such OpenACC support for proprietary OpenMP compilers. However, a known issue with Clang is that its AST, the source-level representation, was designed to be immutable. Moreover, the AST represents the source after preprocessor expansions, which harm readability and can prevent compilation with other compilers. Finally, sophisticated analyses and optimizations are critical for lowering OpenACC's descriptive language to the more prescriptive language of OpenMP, but these are best implemented at the level of LLVM IR not the Clang AST.

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

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

Solution Strategy

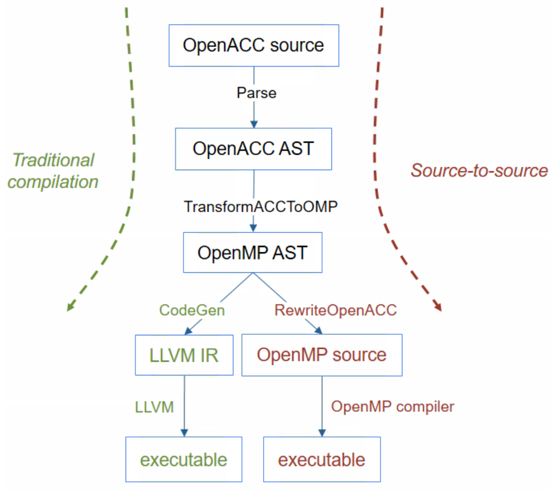
1. A key Clacc design feature is lowering OpenACC to OpenMP. Benefits include:

   (a) By building on Clang and LLVM's OpenMP support, it reduces the effort necessary to construct a production-quality OpenACC implementation.

   (b) It enables OpenACC support on OpenMP compilers other than Clang, including proprietary compilers.

   (c) It facilitates repurposing existing OpenMP static analysis and debugging tools for the sake of OpenACC.

   (d) It facilitates porting applications from OpenACC to OpenMP to alleviate the aforementioned concerns about developing applications in OpenACC.



3. To handle Clang's immutable AST, Clacc's design includes a TransformACCToOMP component that reuses a Clang feature called TreeTransform, which was originally designed for C++ template specializations.

4. To avoid preprocessor expansions in source-to-source mode, Clacc includes a RewriteOpenACC component that reuses a Clang feature called Rewrite.

5. To utilize LLVM IR analyses and optimizations, we are investigating ongoing efforts toward a parallel LLVM IR. Clacc could use such an IR as a code generation target for OpenACC, either directly or after translation to OpenMP extensions Clacc would introduce to support OpenACC's descriptive features.

6. To stage our development effort, we are initially implementing Clacc with two simplifications: we are implementing a prescriptive OpenACC interpretation for correct behavior, and we are implementing for C. We will then extend Clacc with necessary analyses for a descriptive interpretation and for C++.

7. To ensure Clacc's successful implementation and eventual acceptance upstream, we continue design discussions with the Clang and LLVM communities via mailing lists and other relevant forums.

8. Throughout Clacc development, we are continuously integrating the latest upstream Clang and LLVM changes, and we are running and extending the Clang and LLVM test suites to detect regressions and incompatibilities. We are also investigating OpenACC benchmarks [70] and validation test suites [69] to ensure correct OpenACC behavior and good performance.

Recent Progress

1. Extended Clacc to support additional features, including various OpenACC directives and clauses related to data motion and OpenACC Profiling Interface features motivated by integration with TAU.

2. Designed and prototyped various OpenMP extensions to support the above, including OMPT extensions.

3. Developed Gitlab CI config for x86_64, ppc64le, and nvptx64, including a Summit node, on ORNL's ExCL cluster.

4. Performed preliminary GPU evaluation using SPEC ACCEL benchmarks.

5. Contributed numerous improvements to Clang and LLVM, including OpenMP and LLVM testing infrastructure improvements, and to the OpenACC specification.

Next Steps

1. Continue to implement Clacc support for critical OpenACC features based on the needs of ECP and other HPC apps and benchmarks, and pursue OpenACC optimizations and C++ support.

2. Continue contributions to upstream Clang and LLVM and to the OpenACC specification.

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

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

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

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

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

2. Enable continuous integration for the fork on various hardware of interests.

3. Enable LLVM ECP related projects to be able to push and test branches.

4. Setup status information for the continuous information results.

Solution Strategy

1. The DOE LLVM repository is setup on GitHub as a fork of the llvm-project main repository also hosted on GitHub. This makes it easier to have a seamless synchronization with the main repository and keep all the GitHub main-fork integrated features.

2. The GitHub repository is automatically mirrored in the GitLab premium instance hosted at ORNL.

3. The continous integration takes advantage of the GitLab CI infrastructure. This infrastructure is available on several machines form the ExCL lab as well as on Summit and Theta.

Recent Progress

1. Fork is setup with an automatic mirroring with the upstream repository. The mirroring is using GitHub Actions.
2. A GitLab premium instance is running at ORNL and mirrors automatically the GitHub repository. The base continuous integration is running nightly for the main branch of the repository on ExCL machines (Kold and Leconte).

Next Steps

1. Add continuous integration on more hardware (AMD explorer node in ExCL, Summit and Theta)
2. Enhance the continuous integration with additional LLVM sub-projects.
3. Add test-suite to the CI (e.g. SOLLVE validation test-suite).

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

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

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

Despite the importance of OpenACC, the only non-academic open-source OpenACC compiler cited by the OpenACC website is GCC [68]. However, GCC has lagged behind commercial compilers, such as PGI's, in providing production-quality support for the latest OpenACC specifications [69]. Moreover, GCC is known within the compiler community to be challenging to extend and, especially within the DOE, is losing favor to Clang and LLVM for new compiler research and development efforts.

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

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

Key Challenges

1. **OpenACC Support:** Developing production-quality, standards-conforming OpenACC compiler and runtime support is a large undertaking. Complicating that undertaking further is the need for optimization strategies that are competitive with existing commercial compilers, such as PGI's, which have been developed over many years since before the conception of the OpenACC standard.
2. **MLIR:** Flang and the OpenACC support for it rely on the MLIR project for the intermediate representation. MLIR has been upstreamed to the core LLVM project in early 2020 and it is still actively under development. Flang will be the first core frontend relying on MLIR.
3. **Runtime:** LLVM does not include an OpenACC runtime but only one for OpenMP at the moment. This runtime can be generalized to support missing OpenACC features. This generalization needs to be accepted by the current OpenMP community.

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

Solution Strategy

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

Recent Progress

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

Next Steps

1. Continue the definition of the OpenACC MLIR dialect and complete the lowering to it.
2. Start working on the runtime support for OpenACC.
3. Work on the lowering from MLIR to LLVM IR and runtime call.

4.2.15 WBS 2.3.2.10 ***PROTEAS-TUNE: Autotuning***

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

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

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

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

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

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

Recent Progress We have pursued the following main activities this year:

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

SuRF Supporting Autotuning Search Recently, we developed stopping criterion based on local convergence and expected improvement over time. This allows the search to terminate in shorter computation time. Currently, we are expanding the search for multinode autotuning where each evaluation spans multiple nodes. In the past year, we have used SuRF to perform autotuning search on pragmas, including loop transformations and OpenMP pragmas. Most recently, we are using SuRF to refine descriptive OpenMP pragmas such as `# pragma omp loop` to derive prescriptive pragmas for CPU and GPU mapping of code. For this purpose, we refined SuRF to use a python library that supports expressing tree-structured search spaces, including dynamic trees. We have demonstrated that this approach can achieve performance portability across CPU and GPU using OpenMP. We also published a paper on using autotuning to drive loop transformation decisions.

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

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

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

Brick Library: We developed a code generator for the Brick Data Layout library for stencils that is performance-portable across CPU and GPU architectures, and addresses the needs of modern multi-stencil and high-order stencil computations. The key components of our approach that lead to performance portability are (1) a fine-grained brick data layout designed to exploit the inherent multidimensional spatial locality common to stencil computations; (2) vector code generation that can either target wide SIMD CPU instructions sets such as AVX-512 and SIMT threads on GPUs; and, (3) integration with autotuning framework to apply architecture-specific tuning. For a range of stencil computations, we show that it achieves high performance for both the Intel Knights Landing (Xeon Phi) CPU, and the NVIDIA GPUs [71, 72]. This year we extended the library in multiple ways. We show that the indirection in the brick data layout permits distinct physical and logical data layouts; we can therefore store the bricks in memory to reduce the data movement of packing and unpacking during cross-node communication. We have demonstrated strong scaling by reducing communication time.

Next Steps We will continue to work with ECP application teams to integrate our tools with their efforts. In particular, we are integrating bricks into the Proto system, used in subsurface flows.

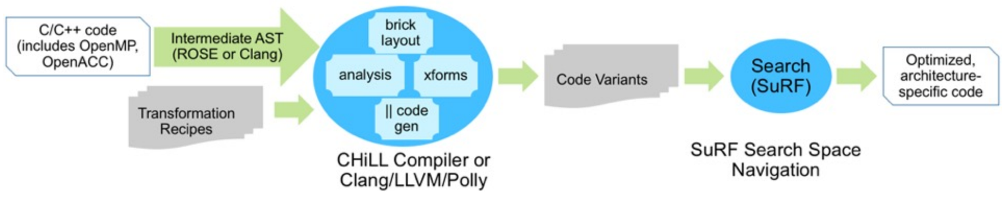

C/C++ code (includes OpenMP, OpenACC)
Intermediate AST (ROSE or Clang)
Transformation Recipes
brick layout
analysis
xforms
|| code gen
CHiLL Compiler or Clang/LLVM/Polly
Code Variants
Search (SuRF)
SuRF Search Space Navigation
Optimized, architecture-specific code

Figure 37: Y-TUNE Solution Approach.

*4.2.16 WBS 2.3.2.10 **PROTEAS-TUNE - Bricks***

Overview We have developed a source-to-source stencil framework ("texttttBricks") to address the growing gap between memory and computational performance on pre-exascale systems. The approach uses standard `C++` code to express stencil loops, then transforms the code to use a different memory alignment and ghost zone region to optimize memory and communication performance specifically for that stencil kernel [71, 72, 73]. This also provides an opportunity to inject architecture-specific code transformations, that can take advantage of SIMD/SIMT, threading, virtual memory layouts, and the variety of parameters that are needed to tune for optimal performance. This is a powerful paradigm for having both a correct legal code base using standard tools and, in combination with the autotuning tools previously described, the ability to achieve portable performance on many different platforms with minimal, auto-generated code transformations.

Key Challenges `Bricks` require three primary ingredients for performance portability:
(1) *Stencil kernel metadata* - including stencil radius, dimensionality, neighbor dependencies, and other memory access patterns. For example, for communication the optimal layout depends on the extent of stencil corner coupling and symmetry or reuse.
(2) *Transformation profitability model* - based on the architecture characteristics and benchmarks, determining what transformations could improve overall throughput, not just maximize flops or bytes moved. This can also be explored using roofline models, auto-tuning, communication-avoiding techniques, etc.
(3) *Back-end optimizations and benchmarks* - knowing what architecture-specific transformations achieve the best roofline performance, and how to isolate and compare those with a known benchmark problem. For example, if there are special OS or hardware capabilities, like vectorized *shuffle*, or memory *mmap* or *prefetch*, that are required to obtain peak performance.

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

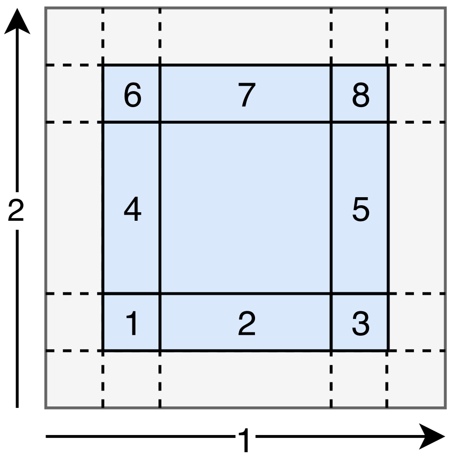


Figure 38: `Bricks` can be used to map memory onto regions that minimize ghost zone packing and MPI message count (2D example).

Recent Progress For optimization of MPI-based communication on exascale proxy systems, we identified several stencil kernels from applications that leverage the tuned kernels from previous milestones, and evaluated their strong scaling on Theta (Intel KNL) and Summit (NVIDIA V100 GPUs). For most stencil codes, strong scaling is limited by the communication of "ghost zone" values or exchanged between processors and nodes, which is required for iterative algorithms or time integrators. Because each MPI rank has one or more subdomains with different layouts in memory, this involves "packing" before sending – copying a subset of local arrays into an MPI message buffer – and then "unpacking" (copying buffers to array subset) after receiving data. This involves both strided memory access and accessing the same array as different messages are sent or received. These operations on the ghost zone "skin" can introduce significant latency and is a blocking operation that, in the limit of strong scaling, can't be hidden by overlapping communication and computation. We have used our `Bricks` source-to-source transformation technique to eliminate the cost of MPI packing/unpacking on CPUs and GPUs, which improves strong scaling for block semi-structured applications, is performance-portable, and is directly relevant to applications with many DoF's per grid point (such as in combustion and multi-physics codes). We have also introduced a novel technique on CPU to significantly reduce the number of messages, using an indirect mapping of memory to MPI buffers.

Next Steps For FY20, we are extending `Bricks` to other application patterns, including block-structured AMR, chemistry kernels, and systems of (non-)linear solvers. For these, the primary focus will be on investigating the profitability models, code transformations, and auto-tuning the kernels. As more diverse architectures and benchmarks become available within the ECP program (AMD GPU, Intel GPU, NVIDIA Turing), we will develop transformations that provide better performance portability. We will be building up to portable *application* performance and load balancing; this is a complex trade-off between all kernels in a given code, and will be very application- and architecture-dependent.

*4.2.17 WBS 2.3.2.10 **PROTEAS-TUNE - TAU Performance System***

Overview The TAU Performance System is a versatile profiling and tracing toolkit that supports performance instrumentation, measurement, and analysis. It is a robust, portable, and scalable performance tool for use in parallel programs and systems over several technology generations. It is a ubiquitous performance tool suite for shared-memory and message-passing parallel applications written in C++, C, Fortran, Java, Python, UPC, and Chapel. In the PROTEAS project, TAU is being extended to support compiler-based instrumentation for the LLVM C, C++, and Fortran compilers using higher-level intermediate language representation. TAU is also targeting support for performance evaluation of directive based compilation solutions using OpenARC and it will support comprehensive performance evaluation of NVM based HPC systems. Through these and other efforts, our objective to better support parallel runtime systems such as OpenMP, OpenACC, Kokkos, ROCm, and CUDA in TAU. Figures 39 and 40 give examples of using TAU's parallel profile analysis tool, ParaProf.

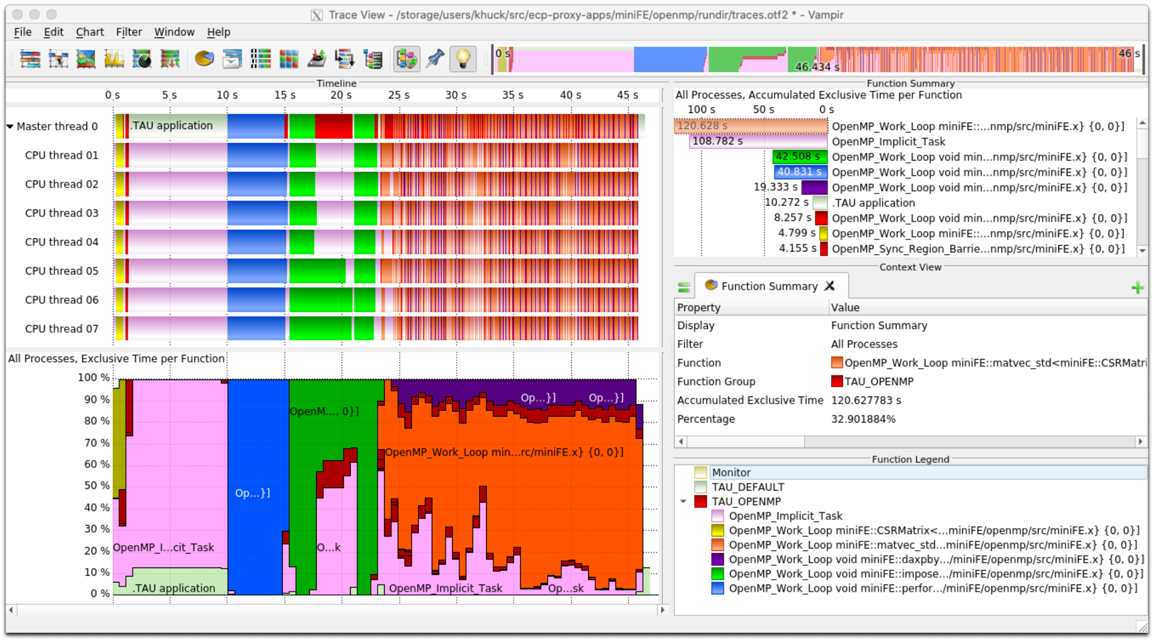
Key Challenges Scalable Heterogeneous Computing (SHC) platforms are gaining popularity, but it is becoming more and more complex to program these systems effectively and to evaluate their performance at scale. Performance engineering of applications must take into account multi-layered language and runtime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

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

Recent Progress

1. **Updated CUDA support** Added preliminary support in TAU for NVIDIA A100 GPUs with support for CUDA 11.

2. **Updated OpenMP support** Updated OMPT support to OpenMP 5.0, tested with ECP Proxy applications miniFE and miniQMC as shown in the Vampir [74] trace viewer in Figure 39.


Trace View - /storage/users/khuck/src/ecp-proxy-apps/miniFE/openmp/rundir/traces.otf2 * - Vampir
File Edit Chart Filter Window Help
Timeline
Master thread 0
CPU thread 01
CPU thread 02
CPU thread 03
CPU thread 04
CPU thread 05
CPU thread 06
CPU thread 07
.TAU application
All Processes, Exclusive Time per Function
OpenMP_I...cit_Task
OpenMP_Work_Loop min...rc/miniFE.x} {0, 0}]
OpenMP_Implicit_Task
Function Summary
Context View
Function Legend

All Processes, Accumulated Exclusive Time per Function

120.628 s	OpenMP_Work_Loop miniFE::...nmp/src/miniFE.x} {0, 0}]
108.782 s	OpenMP_Implicit_Task
42.508 s	OpenMP_Work_Loop void min...nmp/src/miniFE.x} {0, 0}]
40.831 s	OpenMP_Work_Loop void min...nmp/src/miniFE.x} {0, 0}]
19.333 s	OpenMP_Work_Loop void min...nmp/src/miniFE.x} {0, 0}]
10.272 s	.TAU application
8.257 s	OpenMP_Work_Loop void min...nmp/src/miniFE.x} {0, 0}]
4.799 s	OpenMP_Work_Loop miniFE::...nmp/src/miniFE.x} {0, 0}]
4.155 s	OpenMP_Sync_Region_Barrie...nmp/src/miniFE.x} {0, 0}]

Property	Value
Display	Function Summary
Filter	All Processes
Function	OpenMP_Work_Loop miniFE::matvec_std<miniFE::CSRMatri
Function Group	TAU_OPENMP
Accumulated Exclusive Time	120.627783 s
Percentage	32.901884%

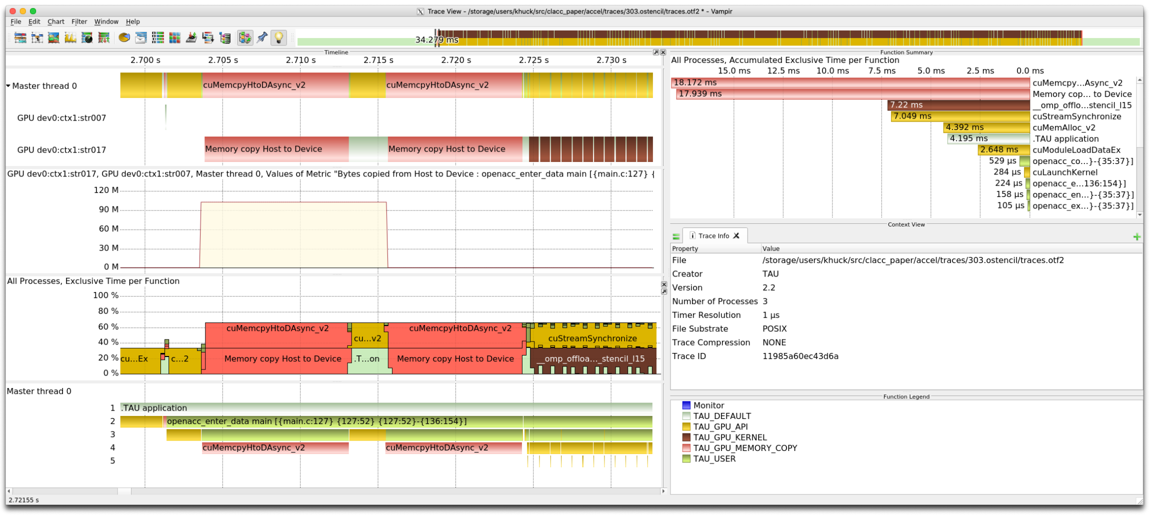
- Monitor
- TAU_DEFAULT
- TAU_OPENMP
  - OpenMP_Implicit_Task
  - OpenMP_Work_Loop miniFE::CSRMatrix<...miniFE/openmp/src/miniFE.x} {0, 0}]
  - OpenMP_Work_Loop miniFE::matvec_std...miniFE/openmp/src/miniFE.x} {0, 0}]
  - OpenMP_Work_Loop void miniFE::daxpby.../miniFE/openmp/src/miniFE.x} {0, 0}]
  - OpenMP_Work_Loop void miniFE::impose.../miniFE/openmp/src/miniFE.x} {0, 0}]
  - OpenMP_Work_Loop void miniFE::perfor.../miniFE/openmp/src/miniFE.x} {0, 0}]

Figure 39: TAU was used to collect profiles and traces of ECP proxy applications like miniFE (trace shown in Vampir), observing OpenMP parallel regions, loops and synchronization without application instrumentation.

3. **Clacc support** Implemented/updated profiling support for OpenACC events provided by the Clacc compiler as shown in the Vampir trace viewer in Figure 40.
4. **HIP** Added support for AMD GPUs with ROCm 3.3.
5. **CODAR** Updated TAU plugin for streaming profile and trace output to ADIOS2 for realtime application monitoring. Integrated with Chimbuko framework for runtime trace analysis, demonstrated with XGC on Summit using 768 MPI ranks. (publication accepted to ISAV Workshop @SC20)
6. **E4S** Integrated TAU in E4S to support AMD GPUs. Both Docker and Singularity images posted on E4S.io website include TAU with support for NVIDIA and AMD GPUs.
7. **Kokkos** Updated support for Kokkos profiling interface in TAU (publication accepted to ProTools Workshop @SC20).
8. **LLVM Instrumentation** Implemented an LLVM module for selective instrumentation of C/C++ using TAU, tested with LLVM versions 6 through 12 and Clacc.
9. **CCAMP** Extended OpenACC and OpenMP interoperable framework, (publication to be presented at SC20).

Next Steps

1. **CUDA Enhancements** Implement new Profiling API and Perfworks Metrics API for CUDA/CUPTI 10+.
2. **OpenMP and OpenACC Enhancements** Explore and implement prototype measurement for OpenMP and OpenACC regions executed on target devices.
3. **New Architectures** We plan to support Intel OneAPI with Level Zero and the HPE Cray platform with AMD GPUs.



All Processes, Accumulated Exclusive Time per Function

18.172 ms	cuMemcpy...Async_v2
17.939 ms	Memory cop... to Device
7.22 ms	__omp_offlo...stencil_l15
7.049 ms	cuStreamSynchronize
4.392 ms	cuMemAlloc_v2
4.195 ms	.TAU application
2.648 ms	cuModuleLoadDataEx
529 µs	openacc_co...}-{35:37}]
284 µs	cuLaunchKernel
224 µs	openacc_e...136:154}]
158 µs	openacc_en...}-{35:37}]
105 µs	openacc_ex...}-{35:37}]

Trace Info

Property	Value
File	/storage/users/khuck/src/clacc_paper/accel/traces/303.ostencil/traces.otf2
Creator	TAU
Version	2.2
Number of Processes	3
Timer Resolution	1 µs
File Substrate	POSIX
Trace Compression	NONE
Trace ID	11985a60ec43d6a

Figure 40: TAU was used to collect profiles and traces of OpenACC benchmarks (303.stencil trace shown in Vampir), observing OpenACC regions and device offload events without application instrumentation.

4. **Outreach** Continued outreach activities to demonstrate comprehensive performance evaluation support in TAU for OpenARC, OpenACC, LLVM compiler-based instrumentation, CUDA, Kokkos, ROCm, and NVM based programming frameworks for SHC platforms.

5. **E4S** Continued integration of TAU and PROTEAS-TUNE projects in the E4S.

6. **LLVM Instrumentation** Add Fortran support for LLVM selective instrumentation module, add OpenACC profiling support for F18.

7. **TAU Instrumentation** Modernize TAU source-to-source auto-instrumentation support in TAU for C++ by replacing current parser front-end with LLVM based solution.

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

Overview Papyrus is a programming system that provides features for scalable, aggregate, persistent memory in an extreme-scale system for typical HPC usage scenarios. Papyrus provides a portable and scalable programming interface to access and manage parallel data structures on the distributed NVM storage. Papyrus allows the programmers to exploit large aggregate NVM space in the system without handling complex communication, synchronization, replication, and consistency models. Papyrus consists of three components, virtual file system (VFS) [60], C++ template container library (TCL) [60], and key-value store (KV) [61]. (1) PapyrusVFS provides a uniform aggregate NVM storage image for the different types of NVM architectures. It presents an illusion of a single large NVM storage for all NVM devices available in the distributed system. Unlike other traditional kernel-level VFSs, PapyrusVFS is a lightweight user-level VFS, which is provided as a library so that applications can link to or dynamically load it. PapyrusVFS implements a subset of POSIX API related to file I/O. (2) PapyrusTCL provides a high-level container programming interface whose data elements can be distributed to multiple NVM nodes. PapyrusTCL provides three containers, including map, vector, and matrix, implemented as C++ templates. PapyrusTCL is built on top of PapyrusVFS. This enables PapyrusTCL to be decoupled from a specific NVM architecture and to present a high-level programming interface whose data elements are distributed across multiple NVM nodes transparently. (3) PapyrusKV is a novel embedded KVS implemented specifically for HPC architectures and applications to provide scalability, replication, consistency, and high performance, and so that they can be customized by the application. It stores keys and values in arbitrary byte arrays across multiple NVMs. PapyrusKV provides configurable consistency technique controlled by the application during the program execution dynamically to meet application-specific requirements and/or needs. It also supports fault tolerance and streamlined workflow by leveraging NVM's persistence property.

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

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

Recent Progress

1. **Data compression and encryption** Added data compression and encryption to Papyrus [75]. Our compression technique exploits deep memory hierarchy in an HPC system to achieve both storage reduction and performance improvement. Our encryption technique provides a practical level of security and enables sharing of sensitive data securely in complex scientific workflows with nearly imperceptible cost.

2. **Redesign** Redesigned and optimized Papyrus to support multidimensional tables.

3. **Summit** Performed preliminary evaluation on OLCF's Summit supercomputer.

Next Steps

1. **Versioning** Versioning can be used to provide new levels of reliability and performance optimization. We will design and implement versioning in Papyrus.

2. **Performance optimization** New APIs and hardware support is being developed for NVM technologies; we are implementing optimizations in Papyrus to take advantage of these advances.

4.2.19 SOLLVE

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

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

The project is developing a verification and validation (V&V) suite to assess implementations and enable evaluations by DOE facilities. It is constructing a high-quality, robust OpenMP implementation based on the LLVM compiler. SOLLVE plays a critical role in specifying, implementing, promoting, and deploying functionality that will enable ECP application developers to reach their goals using OpenMP.

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

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

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

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

1. **Application requirements** Ongoing in-depth interactions with selected ECP application teams have resulted in a list of required extensions, some of which have been met by the recent 5.0 specification. New needs are being identified. This work informs all other project activities by producing use cases, detailed feedback and example codes. It moreover contributes to the OpenMP Examples document.

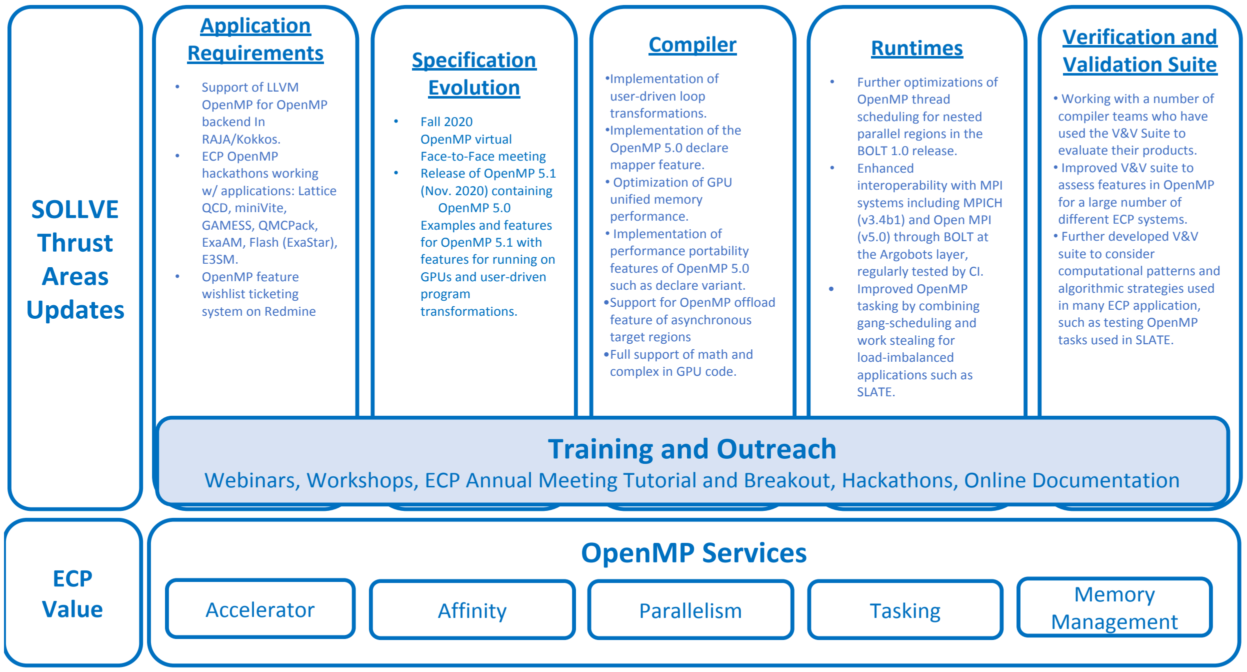
2. **OpenMP specification evolution** Members of the SOLLVE project are active participants in the OpenMP Language committee. The project creates early prototypes for new features based on ECP use cases, develops concrete proposals and submits them for standardization. Several proposed features were included in OpenMP 5.0, ratified November 2018. More are under development for version 5.1.

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

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

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

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



SOLLVE Thrust Areas Updates

Application Requirements

- Support of LLVM OpenMP for OpenMP backend In RAJA/Kokkos.
- ECP OpenMP hackathons working w/ applications: Lattice QCD, miniVite, GAMESS, QMCPack, ExaAM, Flash (ExaStar), E3SM.
- OpenMP feature wishlist ticketing system on Redmine

Specification Evolution

- Fall 2020 OpenMP virtual Face-to-Face meeting
- Release of OpenMP 5.1 (Nov. 2020) containing OpenMP 5.0 Examples and features for OpenMP 5.1 with features for running on GPUs and user-driven program transformations.

Compiler

- Implementation of user-driven loop transformations.
- Implementation of the OpenMP 5.0 declare mapper feature.
- Optimization of GPU unified memory performance.
- Implementation of performance portability features of OpenMP 5.0 such as declare variant.
- Support for OpenMP offload feature of asynchronous target regions
- Full support of math and complex in GPU code.

Runtimes

- Further optimizations of OpenMP thread scheduling for nested parallel regions in the BOLT 1.0 release.
- Enhanced interoperability with MPI systems including MPICH (v3.4b1) and Open MPI (v5.0) through BOLT at the Argobots layer, regularly tested by CI.
- Improved OpenMP tasking by combining gang-scheduling and work stealing for load-imbalanced applications such as SLATE.

Verification and Validation Suite

- Working with a number of compiler teams who have used the V&V Suite to evaluate their products.
- Improved V&V suite to assess features in OpenMP for a large number of different ECP systems.
- Further developed V&V suite to consider computational patterns and algorithmic strategies used in many ECP application, such as testing OpenMP tasks used in SLATE.

Training and Outreach

Webinars, Workshops, ECP Annual Meeting Tutorial and Breakout, Hackathons, Online Documentation

ECP Value

OpenMP Services

Accelerator | Affinity | Parallelism | Tasking | Memory Management

Figure 41: SOLLVE thrust area updates

Recent Progress Figure 41 shows the latest progress on the 5 core SOLLVE thrust areas. The **training and outreach** activity is a cross-cutting effort which is supported by resources from SOLLVE and ECP Broader Engagement, with contributions by external collaborators, notably Lawrence Berkeley National Laboratory. A number of articles have also been published as part of the SOLLVE effort [76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 88].

Next Steps The following next steps are planned:

- Applications: Continue to interact with ECP applications teams, evaluate implementations of new features and explore new requirements; identify best practices for the use of OpenMP on accelerators;
- OpenMP specification: Continue work toward the next version of the standard via ECP-motivated feature development and participation in the OpenMP Language Committee: version 5.1 is underway and due for release November 2020;
- LLVM compiler: Improve performance of device offloading and optimize generation of code within target devices; generalize to enable reuse across multiple offloading architectures; develop infrastructure to support integration of Fortran front end; increase parallel region performance; implementation of OpenMP 5.1 loop transformations;
- OpenMP runtime: Provide support for 5.0 spec; address broader set of interoperability challenges including MPI+OpenMP codes using BOLT; address advanced tasking requirements;
- V&V suite: Continue expanding the coverage of the V&V Suite, with focus on 5.0 features and additions and corrections to 4.5 tests as OpenMP implementations mature; expand Fortran tests; work with ARB Examples Committee; improve ALCF toolchains; more vendor interactions.

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

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

Key Challenges Several user-level threading and tasking models have been proposed in the past to address the shortcomings of OS-level threads, primarily with respect to cost and flexibility. Their lightweight nature and flexible generic interface play an important role at managing efficiently the massive concurrency expected at the Exascale level. Existing user-level threading and tasking models, however, are either too specific to applications or architectures or are not powerful or flexible. Existing runtimes tailored for generic use [93, 94, 95, 96, 97, 98, 99, 100, 101] are suitable as common frameworks to facilitate portability and interoperability but offer insufficient flexibility to efficiently capture higher-level abstractions, while specialized runtimes [102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112] are tailored to specific environment.

Solution Strategy Argobots offers a carefully designed execution model that balances generality of functionality with providing a rich set of controls to allow specialization by end users or high-level programming models [113]. Delivering high performance in Argobots while providing a rich set of capabilities is achieved by heavily optimizing critical paths as well as by exposing configuration knobs and a rich API, which allow users to trim unnecessary costs. Furthermore, Argobots honors high degrees of expressibility through the following three key aspects:

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

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

3. Allowing full control over *work unit management*. Users can freely manage *scheduling* and mapping of work units to ESs through *thread pool* management, and thus achieving the desired behavior. Figure 42 illustrates the various building blocks in the Argobots framework and the interactions between them to build a hypothetical system.

Recent Progress Threading overheads are crucial for fine-grained parallel applications and runtimes running in massively parallel environments. We have found that the timing of yield operations highly affects the performance of lightweight threads [114], but other factors remained unexplored. We further optimized fork-join overheads by exploring new threading methods with respect to stack allocation timing and scheduling policies and a wider range of modern hardware architectures. Our evaluation shows that our child-first scheduling yields promising results for deep and narrow recursive task-parallel programs while the parent-first scheduling is good for flat parallelism. Our study helps users and application developers choose the best threading methods that fit their hardware architectures and application workloads and maximize the scalability [115].

Integration with other runtime systems is fundamentally important for the Argobots project. BOLT, a SOLLVE OpenMP runtime over Argobots [34], is one of the most successful parallel programming systems using Argobots. Our enhancements of Argobots threads lowers the cost of OpenMP threading and tasking. The Argobots project continues to improve interoperability with communication layers such as MPI runtimes (e.g., MPICH and Open MPI) and Margo, a Mercury RPC over Argobots. To help their performance analysis, our latest Argobots 1.1a1 release includes a lightweight yet powerful profiling interface, which helps runtime developers pinpointing a performance problem in these systems. I/O service is one of the most important

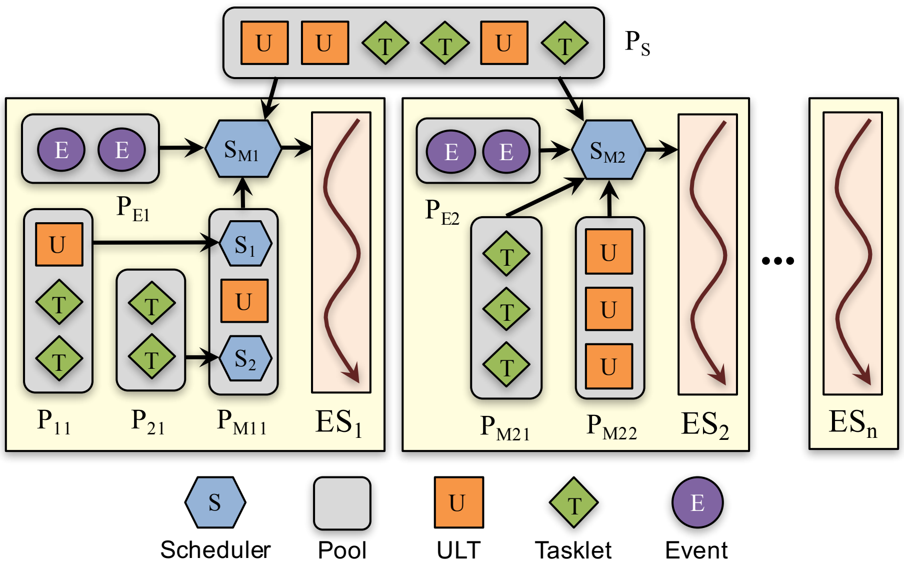


Figure 42: Argobots execution model

application areas for Argobots. Intel DAOS, a next- generation high-performance storage system developed by Intel, uses Argobots to efficiently handle asynchronous I/O messages. We are working together to improve Argobots by providing a better debugging interface such as stack dump features. Our CI testing has been extended for various CPU architectures, operating systems, and compilers to cover most of the DOE HPC platforms. Thanks to our CI, Argobots 1.1a1 works on major UNIX-based platforms including Ubuntu, FreeBSD, CentOS, macOS, and Solaris. Argobots supports most CPU architectures with special optimizations for Intel/AMD x86/64, ARMv8-A, and POWER 8 and 9. Argobots can be compiled with numerous C compilers including GCC, Clang, ICC (Intel), XLC (IBM), PGCC (PGI), Solaris Studio (Oracle), and Arm C Compiler for HPC (ARM).

The innovative design and implementation of Argobots are highly recognized. Argobots was named a finalist for the 2020 R&D 100 Awards. The prestigious R&D 100 competition, sponsored by R&D Magazine, recognizes the 100 most innovative technologies of the previous year. Argobots, a lightweight and highly flexible multithreading framework, was chosen as a finalist for the 2020 R&D 100 Awards.

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

1. Further integration with other applications and runtimes including MPI runtimes including MPICH and Open MPI. In collaboration with MPICH and Open MPI developers, we will further optimize their Argobots interoperability layers by utilizing user-level threading techniques.

2. Enhanced interoperability of multiple components that are not aware of Argobots. Unfortunately, not all applications are written for lightweight ULTs; some programs suffer from core starvation and, in the worst case, deadlocks if they are running on Argobots. To address this issue, we are investigating an approach that is as lightweight as the current Argobots ULTs while it has the OS-implicit preemption functionality that traditional OS-level threads have.

4.2.21 WBS 2.3.2.11 BOLT: Lightning Fast OpenMP

Overview OpenMP is central for several applications that target Exascale, including ECP applications, to exploit on-node computational resources. Unfortunately, current production OpenMP runtimes, such as those that ship with Intel and GNU compilers, are inadequate for the massive and fine-grained concurrency expected at the Exascale level. These runtimes rely on heavy-handed OS-level threading strategies that incur significant overheads at fine-grained levels and exacerbate interoperability issues between OpenMP and internode programming systems, such as MPI and OpenSHMEM. BOLT is a production quality OpenMP runtime (called BOLT) which has been developed within the SOLLVE project to address this issue by leveraging user-level threads instead of OS-level threads (e.g., Pthreads). Due to their lightweight nature, managing and scheduling user-level threads incurs significantly less overheads. Furthermore, interoperability between BOLT and internode programming systems opens up new optimization opportunities by promoting asynchrony and reducing hardware synchronization (atomics and memory barriers). Initial studies on this proposal can be found in [116, 117, 118]. This report briefly summarizes the issues in OpenMP runtimes that rely on OS-level threading, describes BOLT as the solution to this challenge, the current status in the BOLT effort, and the next steps for further improvements.

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

Solution Strategy Both fine-grained parallelism and interoperability issues suffer from the heavy nature of working at the level of OS threads. Our solution to both challenges leverages user-level threads. Using user-level threads as the underlying threading layer for the OpenMP runtime offers a significantly better trade-off between high concurrency and thread management overheads. This allows users to generate fine-grained concurrency and oversubscription without worrying about the performance collapse that is observed in current OpenMP runtimes. Our OpenMP runtime, BOLT, is derived from the LLVM OpenMP runtime and leverages Argobots, a highly optimized lightweight threading library, as its underlying threading layer. OpenMP threads and tasks are spawned as Argobots work units and nested parallel regions are managed through an efficient work-stealing scheduler. Furthermore, new compiler hints and runtime optimizations have been developed to allow reducing thread management overheads even further [114, 115]. Interoperability improvements have also been demonstrated by having BOLT interoperate with an MPI libraries through the Argobots threading layer rather than OS-level threads. Results showed that this approach allows better communication progress and outperforms the traditional Pthreads-level interaction [113].

Recent Progress We improved the interoperability of BOLT with various MPI systems via lightweight threads, Argobots. Since most MPI runtimes including MPICH, Open MPI, and production MPI implementations that are derived from either MPICH or Open MPI assume OS-level threads as "Thread" in MPI+Thread, lightweight OpenMP runtimes based on lightweight threads failed to interoperate well with existing MPI systems. To address this issue, we have implemented an abstracted threading layer for lightweight threads in these MPI runtimes so that users can choose OpenMP threads and tasks of BOLT as "Thread".

Specifically, we focused on the MPI interoperability with the most widely used open-source MPI implementations: MPICH and Open MPI. For MPICH, we fixed a few bugs regarding synchronization mechanisms, which will be included in the MPICH 3.4 release. In collaboration with the Open MPI researchers and the Qthreads researchers at Sandia and Los Alamos National Laboratories, we implemented a new thread abstraction layer for generic threading runtime support in Open MPI using the Opal Modular Component

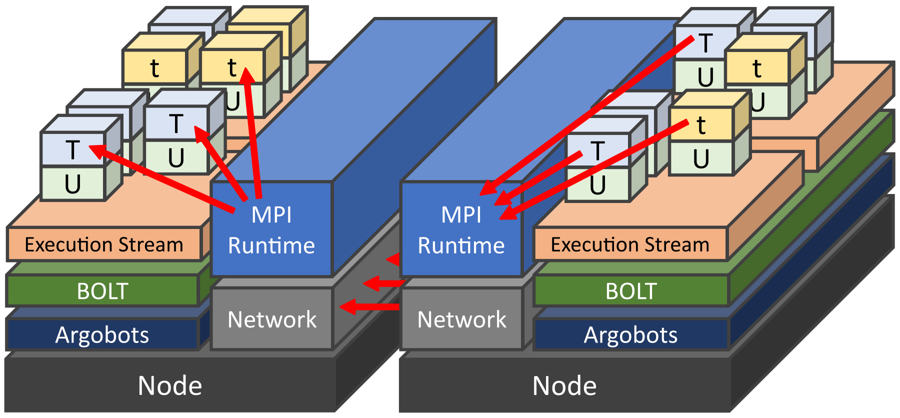


Figure 43: MPI+Threads interoperability of BOLT. OpenMP threads and tasks in BOLT interact MPI implementations via the Argobots layer.

Architecture. This highly abstracted threading layer has been implemented carefully to minimize the additional overheads. Furthermore, this Opal architecture partly allows the programmer to choose an underlying threading library at compile- or run-time to provide flexibility. The performance trade-off has been discussed in our work [119]. This Open MPI interoperability improvement will be included in the Open MPI 5.0 release. To ensure availability, we employed a weekly CI testing infrastructure for those MPI systems.

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

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

1. Explores opportunities for utilizing lightweight threads for other optimizations in the context of OpenMP. The main focus of BOLT has been the performance of fine-grained OpenMP threads, so we have not fully explored how BOLT could elevate the performance of other parallel units (e.g., data-dependent tasking and GPU offloading). We are planning to investigate room for optimizations and implement them with evaluation.

2. Investigates the performance with large-scale applications. In order to find potential room for optimizations and evaluate the performance of BOLT in real large-scale workloads, we further investigate other SOLLVE components and ECP applications that can benefit from BOLT. Since most distributed systems rely on MPI for internode communication, we will also work on tighter integration with MPI to optimize the performance with MPI runtimes over BOLT.

[

4.2.22 WBS 2.3.2.12 **Flang**

Overview The Flang project provides an open source Fortran [120] [121] [122] compiler. The project was formally accepted as a component of the LLVM Compiler Infrastructure last year (see http://llvm.org) [123] and has merged portions of its initial codebase into the main LLVM repository as of April 2020. Leveraging LLVM, Flang will provide a cross-platform Fortran solution available to ECP and the broader international LLVM community. The goals of the project include extending support to GPU accelerators and target Exascale systems, and supporting LLVM-based software and tools R&D of interest to a large deployed base of Fortran applications.

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

Key Challenges Today there are several commercially-supported Fortran compilers, typically available on only one or a few platforms. None of these are open source. While the GNU gfortran open source compiler is available on a wide variety of platforms, the source base is not modern LLVM-style C++ and the GPL open source license is not compatible with LLVM, both of which can impact broader community participation and adoption.

The primary challenge of this project is to create a source base with the maturity, features, and performance of proprietary solutions with the cross-platform capability of GNU compilers, and which is licensed and coded in a style that will be embraced by the LLVM community. Additional challenges come from robustly supporting all Fortran language features, programming models, and scalability required for effective use on Exascale systems.

Solution Strategy With the adoption of Flang into the LLVM community, our strategy focuses on building a strong community around it and the development and delivery of a solid, alternative Fortran compiler for DOE's Exascale platforms. It is critical that we be good shepherds within the broad LLVM community to successfully establish and grow a vibrant community for Flang. This external engagement is in the best interest of ECP as well as the long-term success of Fortran in the LLVM community and the many industry products that rely upon it.

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

Recent Progress After several years of effort and support from NNSA, Flang was successfully *"adopted"* by the LLVM community and has transitioned from a stand-alone git repository to one hosted by the main LLVM project. This represents a significant result and the current code is available via GitHub:

https://github.com/llvm/llvm-project/tree/master/flang

The current capabilities of flang include the full Fortran 2018 standard and OpenMP 5.X syntax and semantics. As part of the development of the parsing and semantic analysis portions of the front-end, over five million lines of Fortran code has been successfully processed. Beyond parsing and semantics, we have been focusing our efforts on creating a Fortran-centric intermediate representation (Fortran IR – "FIR") that leverages recent activities within Google on https://www.blog.google/technology/ai/mlir-accelerating-ai-open-source-infrastructure/ (MLIR) for use with the implementation of FIR. With the development of FIR progressing, we are approaching completion of the first full (sequential) compiler with the full set of F77 capabilities, and implementations of later standards are expected to follow naturally after that.

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

4.3 WBS 2.3.3 MATHEMATICAL LIBRARIES

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

4.3.1 Scope and Requirements

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

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

Advances in mathematical and scientific libraries will be necessary to enable computational science on Exascale systems. Exascale computing promises not only to provide more computational resources enabling higher-fidelity simulations and more demanding studies but also to enable the community to pose new scientific questions. Exascale architectural characteristics introduce new features that algorithms and their implementations will need to address in order to be scalable, efficient, and robust. As a result, it will be necessary to conduct research and development to rethink, reformulate, and develop existing and new methods and deploy them in libraries that can be used by applications to deliver more complete and sophisticated models and provide enhanced predictive simulation and analysis capabilities.

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

4.3.2 Assumptions and Feasibility

Years of DOE investment have led to a diverse and complementary collection of mathematical software, including AMReX, Chombo, hypre, Dakota, DTK, MAGMA, MFEM, PETSc/TAO, PLASMA, ScaLAPACK, SUNDIALS, SuperLU, and Trilinos. This effort is evolving a subset of existing libraries to be performant on Exascale architectures. In addition, research and development is needed into new algorithms whose benefits may be seen only at the extreme scale. Results of preliminary R&D projects indicate that this approach is feasible.

Additionally, ECP will need to rely on a strong, diverse, and persistent base math research program, which is assumed to continue being supported by the DOE-SC ASCR Office. The ECP technical directors will schedule quarterly meetings with the ASCR research program managers to get updates on research results that might meet ECP requirements as well as to inform the program managers of ECP needs in applications and software components.

4.3.3 Objectives

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

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

4.3.4 Plan

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

Initial efforts have focused on identifying core capabilities needed by selected ECP applications, establishing performance baselines of existing implementations on available Petascale and prototype systems, and beginning re-implementation of lower-level capabilities of the libraries and frameworks. Another key activity is collaborating across all projects in the Mathematical Libraries effort to define community policies in order to enable compatibility among complementary software and to provide a foundation for future work on deeper levels of interoperability. Refactoring of higher-level capabilities will be prioritized based on needs of the applications. In time, these efforts will provide demonstrations of parallel performance of algorithms from the mathematical software on pre-Exascale, leadership-class machines (at first on test problems, but eventually in actual applications). The initial efforts also are informing research into advanced exascale-specific numerical algorithms that will be implemented within the libraries and frameworks. In FY20–23, the focus will be on development and tuning for the specific architectures of the selected exascale platforms, in addition to tuning specific features that are critical to ECP applications. The projects will implement their software on the CORAL, NERSC and ACES systems, the pre-Exascale hardwares such as Tulip and Iris, and ultimately on initial Exascale systems, so that functionality, performance, and robustness can be evaluated by the applications teams and other elements of the software stack. Throughout the effort the applications teams and other elements of the software stack will evaluate and provide feedback on their functionality, performance, and robustness. These goals will be evaluated at least yearly based on milestones as well as joint milestone activities shared across the associated software stack activities by Application Development and Hardware and Integration project focus areas.

4.3.5 Risks and Mitigations Strategies

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

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

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

- Exascale architecture characteristics may force a much tighter coupling among the models, discretizations, and solvers employed, causing general-purpose solvers to be too inefficient. The mitigation strategy is to ensure close collaboration with the sub-elements of the Application Development focus area (WBS

2.2) to understand integration and coupling issues. Again, a strong, diverse, well-run, persistent base math research program may provide risk mitigation strategies.

4.3.6 Future Trends

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

In addition, we anticipate that new algorithms based on multi-precision arithmetic will further enable performance improvements on compute devices that are optimized for machine learning workloads, where lower precision can be an order of magnitude faster than double precision. A recent paper [124] surveys the landscape of multi-precision numerical linear algebra algorithms.

For a deeper discussion of the futures of ECP Math Libraries efforts, please consult the paper "Preparing Sparse Solvers for Exascale Computing" [125].

4.3.7 WBS 2.3.3.01 xSDK

Overview The xSDK project is creating a value-added aggregation of DOE math and scientific libraries through the xSDK (Extreme-scale Scientific Software Development Kit) [126], which increases the combined usability, standardization, and interoperability of these libraries as needed by ECP. The project focuses on community development and a commitment to combined success via quality improvement policies, better build infrastructure, and the ability to use diverse, independently developed xSDK libraries in combination to solve large-scale multiphysics and multiscale problems. We are extending xSDK package community policies and developing interoperability layers among numerical libraries in order to improve code quality, access, usability, interoperability, and sustainability. Focus areas are (1) coordinated use of on-node resources, (2) integrated execution (control inversion and adaptive execution strategies), and (3) coordinated and sustainable documentation, testing, packaging, and deployment.

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

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

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

Solution Strategy The xSDK effort has two primary thrusts:

1. **Increased interoperability:** xSDK packages can be built with a single Spack package target. Furthermore, services from one package are accessible to another package.

2. **Increased use of common best practices:** The xSDK has a collection of community policies that set expectations for a package, from best design practices to common look-and-feel.

xSDK interoperability efforts began first with eliminating incompatibilities that prohibited correct compilation and integration of the independently developed libraries. These issues include being able to use a common version of a library by another library. The second, and ongoing phase is increased use of one package's capabilities from another. xSDK community package policies [3, 127] are a set of minimum requirements (including topics of configuring, installing, testing, MPI usage, portability, contact and version information, open source licensing, namespacing, and repository access) that a software package must satisfy in order to be considered xSDK compatible. The designation of xSDK compatibility informs potential users that a package can be easily used with others and makes configuration and installation of xSDK software and other HPC packages as efficient as possible on common platforms, including standard Linux distributions and Mac OS X, as well as on target machines currently available at DOE computing facilities (ALCF, NERSC, and OLCF) and eventually on new Exascale platforms. Community policies for the xSDK promote long-term sustainability and interoperability among packages, as a foundation for supporting complex multiphysics and multiscale ECP applications. In addition, because new xSDK packages will follow the same standard, installation software and package managers (for example, Spack [1]) can easily be extended to install many packages automatically.

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

Recent Progress The xSDK team developed a suite of example codes that demonstrate interoperabilities between select xSDK libraries, xsdk-examples v.0.1.0 [128]. The suite includes a build system and documentation in the subfolders of the codes and can be built with Spack [1]. It provides training for xSDK users on mixed package use. It also serves as test suite and will be included in testing of future xSDK releases. Figure 44 illustrates the xSDK libraries and their interoperabilities represented in the first release.

The xSDK team also released version v.0.6.0 of the xSDK community policies [127]. It includes a new recommended policy on documentation quality. Since the switch from the original xSDK installer to Spack as the xSDK package installer has facilitated the build of the xSDK, the team could simplify policy M1 by merging it with M16 and abandoning the installation policies. In place of the installation policies, Spack variant guidelines have been provided, and a new policy M16 was created to keep the installation policy requirement that xSDK libraries need to have an option to be configured in debug mode.

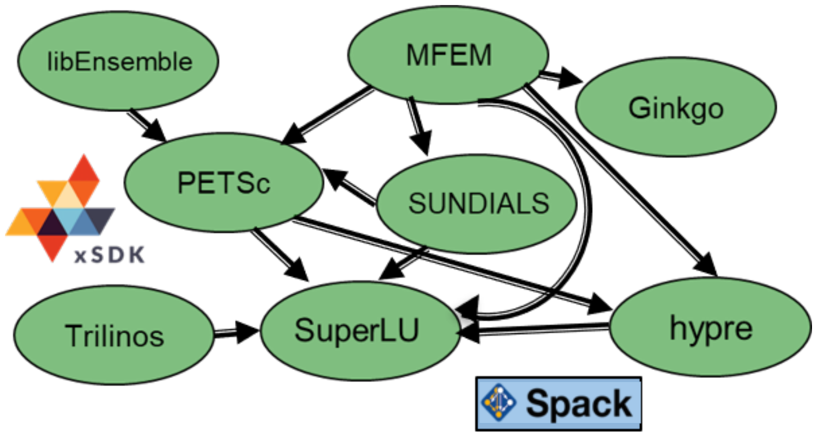


Figure 44: xSDK packages and interoperabilities represented in version v.0.1.0 of the xsdk-examples test suite. A→B indicates that A uses functionalities of B

The first version of the GPTune autotuning software for parameter optimization of HPC codes was released [129]. It was evaluated by tuning several ECP math libraries and applications codes using up to 2048 Cori Haswell cores. GPTune achieved a performance gain of up to 60 percent compared to default parameter settings. It outperformed two state-of-the-art tuners, OpenTuner and HpBandster, up to 2.5, when tuning ScaLAPACK QR.

Next Steps Our next efforts include

- a new xSDK release with two additional math libraries heFFTe and SLATE,
- development of new interoperabilities between xSDK libraries and their inclusion in xsdk-examples,
- enhancing GPTune with new features, such as transfer learning, incorporation of predictive models, and speeding up the internal Gaussian process modeling algorithms,
- design and implementation of a code quality toolkit that automates analyses and activities related to code testing, documentation, and use.

4.3.8 WBS 2.3.3.01 *xSDK Sub-project: multiprecision*

Overview Within the past years, hardware vendors have started designing low precision special function units in response to the demand of the Machine Learning community for high compute power in low precision formats. At the same time, the gap between compute power on the one hand and memory bandwidth on the other hand keeps increasing, making data access and communication prohibitively expensive compared to arithmetic operations. Having the choice between ignoring the hardware trends and continuing the traditional path or adjusting the software stack to the changing hardware designs, the US Exascale Computing Project decided to build a multiprecision focus effort to take on the challenge of designing and engineering novel algorithms capable to exploit the compute power available in low precision and to adjust the communication format to application specific needs. As part of the xSDK project, the multiprecision focus effort is a coordinated effort creating synergies via cross-institutional collaboration.

Key Challenges Generally, there exists a strong relationship between the precision used in arithmetic operations and the accuracy of the computed result. Since scientific applications need to provide high quality output, replacing high precision formats with low precision formats throughout a complete application code is generally not feasible. Instead, to utilize lower precision formats, the underlying numerical algorithms have to be redesigned to employ low precision formats for the most time-consuming parts while preserving high accuracy in the solution. In this context, the arithmetic operations are only one aspect. As the execution time of many scientific applications is dominated by communication and memory access, the algorithm redesign also has to include strategies for compressing data to reduce the pressure on the memory bandwidth. This aspect becomes even more relevant as the arithmetic power continues to grow faster than the memory bandwidth, therewith widening the gap between arithmetic performance and memory performance, see Figure 45.

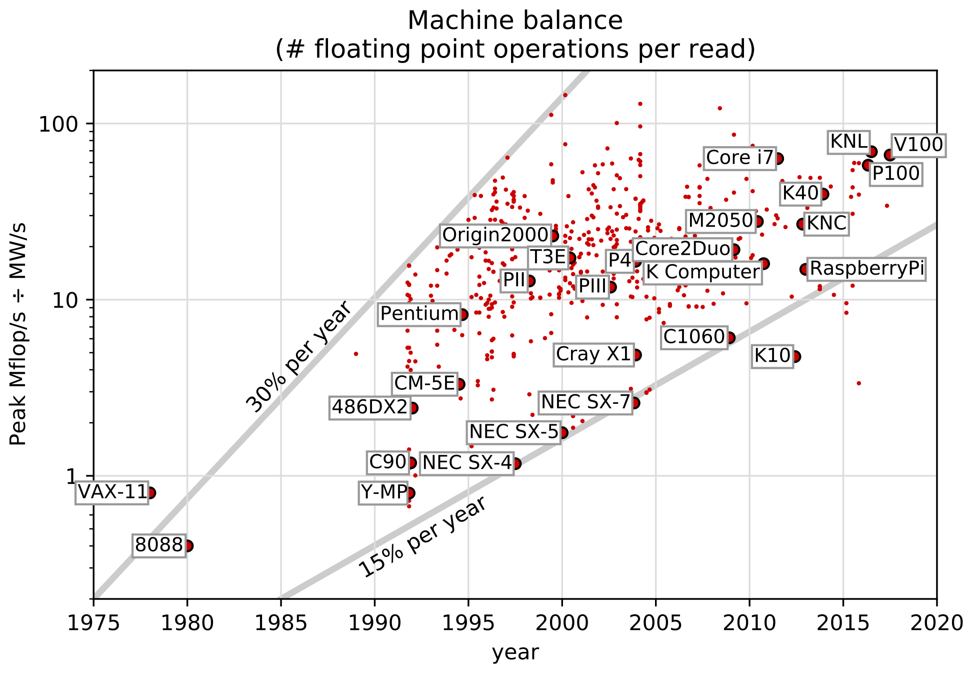


Figure 45: Evolution of the machine balance of processors over different hardware generations.

Solution Strategy In the multiprecision effort, the team assesses current status and functionalities, advances the theoretical knowledge on multiprecision algorithms, designs prototype implementations and multiprecision interoperability layers, deploys production-ready multiprecision algorithms in the xSDK math libraries, ensures multiprecision cross-library interoperability and integrates multiprecision algorithms into

ECP application projects. The long list of project activities is organized in a multi-phase approach: The first stage is dedicated to the exploration of the design space and existing research efforts focusing on low precision, mixed precision, and extended precision. To this end, the effort includes internationally-renown external experts for mixed precision algorithms. In the second stage, the multiprecision effort increasingly focuses on developing production code ready to be used in the ECP application projects. Relevant algorithmic functionality includes sparse linear algebra, multigrid methods, preconditioners, iterative solvers, low-rank approximations, and mixed precision machine learning. On the hardware side, special focus is put on low-precision special function units like NVIDIA's tensor cores that – originally designed for machine learning algorithms – offer about an order of magnitude higher arithmetic performance than conventional fp64 units. In the third phase of the multiprecision effort, the project team will aid the ECP application projects with the adoption of multiprecision functionality and continue to adapt to new hardware technologies.

Recent Progress As a first step, the multiprecision team surveyed the state of the art in terms of mixed precision algorithms, low precision and extended precision algorithms, and the algorithmic needs of the ECP application projects. The results of this landscape assessment are made publicly available as "A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic" [124]. A more compact version of this survey – exclusively focusing on numerical linear algebra – has been submitted as a journal article.

Based on the findings of the multiprecision landscape assessment, we created a set of cross-institutional focus groups that address different algorithms and aspects of the ECP software ecosystem including dense and sparse direct solvers, eigensolvers, Krylov-based iterative solvers, multigrid methods, preconditioners, Fast Fourier Transform, and machine learning technology. Orthogonal to those, we created focus groups on the design on a memory accessor that separates the arithmetic precision from the communication precision, and the efficient realization of multiprecision basic building blocks like sparse matrix vector multiply. In bi-weekly virtual meetings, the progress on the different efforts is presented and discussed as many of these efforts are closely related. As another integral part of the bi-weekly phone calls we established a series of short talks where each meeting is commenced by an invited talk presenting an idea, success story, or progress update on mixed precision functionality to the audience.

Following the idea of decoupling the memory precision from the arithmetic precision to reduce the pressure on the memory bandwidth, the team released an accessor design document detailing the implementation and usage of a memory accessor that compresses data, e.g. by converting to a lower precision format, before invoking memory operations. The document has been made available to the ECP community.

Next Steps Our next efforts include

- the publication of a compacted version of the multiprecision landscape assessement as a scientific journal paper,
- the deployment of the accessor separating memory precision and arithmetic precision in the Ginkgo library with support for AMD GPUs, NVIDIA GPUs, and multicore CPUs,
- the implementation of compressed basis Krylov solvers that utilize the memory accessor to compact the Krylov search directions,
- the advancement of multiprecision capabilities for solvers, preconditioners, and other ECP-relevant kernels in xSDK libraries, including
    - the design and implementation of mixed precision eigensolvers,
    - the research and realization of mixed precision multigrid solvers,
    - the design and implementation of mixed precision sparse factorizations.

4.3.9 WBS 2.3.3.06 *PETSc-TAO*

Overview Algebraic solvers (generally nonlinear solvers that use sparse linear solvers) and integrators form the core computation of many numerical simulations. No scalable "black box" sparse solvers or integrators work for all applications, nor are there single implementations that work well for all problem sizes. Hence, algebraic solver and integrator packages provide a wide variety of algorithms and implementations that can be customized for the application and range of problem sizes. PETSc/TAO [130, 131] is a widely used numerical library for the scalable solution of linear, nonlinear, and variational systems, for integration of ODE/DAE systems and computation of their adjoints, and for numerical optimization. This project focuses on three topics: (1) partially matrix-free scalable solvers to efficiently use many-core and GPU-based systems; (2) reduced synchronization algorithms that can scale to larger concurrency than solvers with synchronization points; and (3) performance and data structure optimizations for all the core data structures to better utilize many-core and GPU-based systems as well as provide scalability to the exascale systems.

The availability of systems with over 100 times the processing power of today's machines compels the utilization of these systems not just for a single "forward solve" (as discussed above), but rather within a tight loop of optimization, sensitivity analysis (SA), and uncertain quantification (UQ). This requires the implementation of a new scalable library for managing a dynamic hierarchical collection of running scalable simulations, where the simulations directly feed results into the optimization, SA, and UQ solvers. This library, which we call libEnsemble, directs the multiple concurrent "function evaluations" through the tight coupling and feedback. This work consist of two parts: (1) the development of libEnsemble; and (2) the development of application-relevant algorithms to utilize libEnsemble.

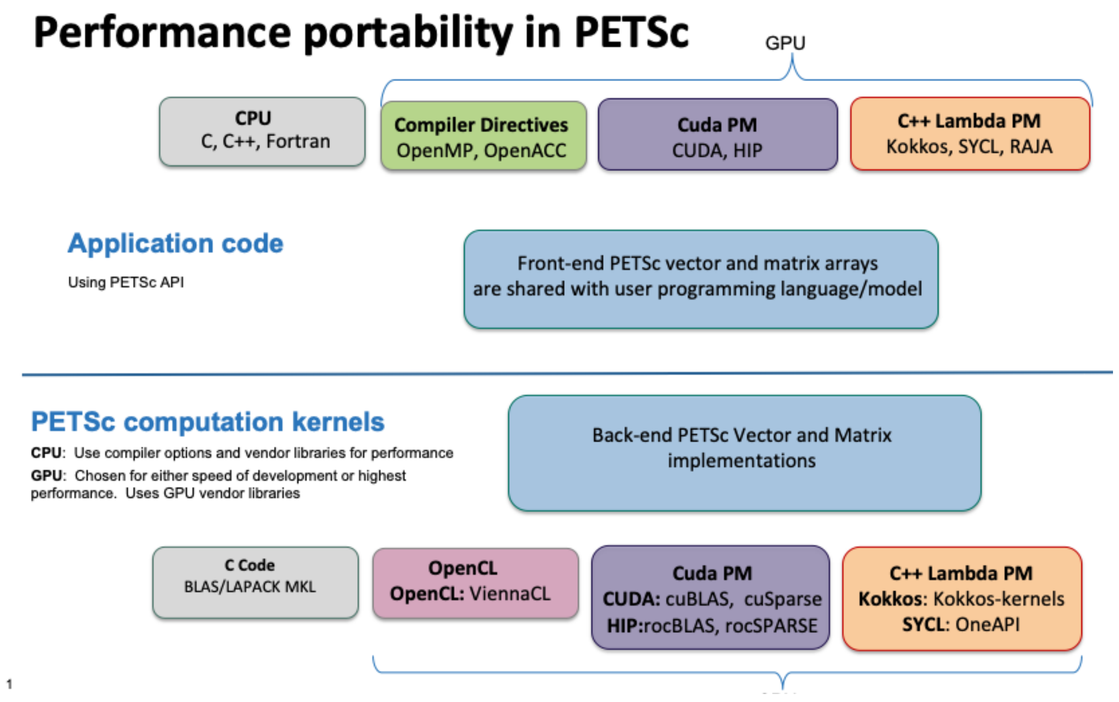
Key Challenges A key challenge for scaling the PETSc/TAO numerical libraries to Exascale systems is that traditional "sparse-matrix-based" techniques for linear, nonlinear, and ODE solvers, as well as optimization algorithms, are memory-bandwidth limited. Another difficulty is that any synchronizations required across all compute units—for example, an inner product or a norm—can dramatically affect the scaling of the solvers. Another challenge is the need to support the variety of accelerators that will be available on the exascale systems and the programming models that application teams use for performance portability.

Running an ensemble of simulations requires a coordination layer that handles load balancing and allows the collection of running simulations to grow and shrink based on feedback. Thus, our libEnsemble library must be able to dynamically start simulations with different parameters, resume simulations to obtain more accurate results, prune running simulations that the solvers determine can no longer provide useful information, monitor the progress of the simulations, and stop failed or hung simulations, and collect data from the individual simulations both while they are running and at the end.

Solution Strategy To address the scalability of the numerical libraries, we implemented new solvers and data structures including: pipeline Krylov methods that delay the use of the results of inner products and norms, allowing overlapping of the reductions and other computation; partially matrix-free solvers using high-order methods that have high floating-point-to-memory-access ratios and good potential to use many-core and GPU-based systems; and in-node optimizations of sparse matrix-matrix products needed by algebraic multigrid to better utilize many-core systems.

Our strategy for coordinating ensemble computations has been to develop libEnsemble to satisfy our needs. This library should not be confused with workflow-based scripting systems; rather it is a library that, through the tight coupling and feedback, directs the multiple concurrent "function evaluations" needed by optimization, SA, and UQ solvers.

Recent Progress In the past year, we have released PETSc/TAO 3.14 (available at `http://www.mcs.anl.gov/petsc`), which features enhanced GPU support. The library now supports CUDA-11 and HIP, along with CUDA-aware MPI, which allows direct communication of data between Summit GPUs, bypassing the previously needed step of first copying the data to the CPU memory. This enhancement reduces the latency of the communication and improves bandwidth. An experimental Kokkos backend for some matrix and vector operations using KokkosKernels was also provided, as one step in the refactoring process to support the variety of accelerators needed for exascale systems and the programming models for performance portability wanted by applications.



Performance portability in PETSc
GPU
CPU
C, C++, Fortran
Compiler Directives
OpenMP, OpenACC
Cuda PM
CUDA, HIP
C++ Lambda PM
Kokkos, SYCL, RAJA
Application code
Using PETSc API
Front-end PETSc vector and matrix arrays are shared with user programming language/model
PETSc computation kernels
CPU: Use compiler options and vendor libraries for performance
GPU: Chosen for either speed of development or highest performance. Uses GPU vendor libraries
Back-end PETSc Vector and Matrix implementations
C Code
BLAS/LAPACK MKL
OpenCL
OpenCL: ViennaCL
Cuda PM
CUDA: cuBLAS, cuSparse
HIP:rocBLAS, rocSPARSE
C++ Lambda PM
Kokkos: Kokkos-kernels
SYCL: OneAPI
1

Figure 46: The improved PETSc/TAO architecture enables users to utilize a variety of programming models for GPUs independently of PETSc's internal programming model.

We have also release libEnsemble 0.7.1 (available at `https://github.com/Libensemble/libensemble`). This release includes new generator functions and examples, changes to become xSDK compatible, and improved testing across available platforms.

Next Steps Our next efforts are:

1. **Performance and application assessment**: We will provide updated performance reports of PETSc/TAO on the architectures available to us. We will work with our applications to assess the usage of our software technologies and our progress toward reaching our impact goals. We will add a libEnsemble guide for function writer users to the documentation and survey the libEnsemble user community.
2. **PETSc/TAO release with full-functionality on available hardware**: We will release a version of PETSc/TAO that fully supports the hardware and software on the architectures available to us. We will begin testing important kernels using different backends and prepare more methods to utilize accelerators.
3. **libEnsemble release with enhanced capabilities**: We will release a version of libEnsemble that implements a method for Bayesian calibration. We will connect libEnsemble to continuous integration tools and demonstrate capabilities.
4. **PETSc/TAO release focused on performance on available hardware**: We will release a version of PETSc/TAO with performance improvements on the architectures available to us. We will continue testing important kernels using different backends and optimize more methods to utilize accelerators.

4.3.10 WBS 2.3.3.07 STRUMPACK-SuperLU

Overview This project will deliver factorization-based sparse solvers encompassing the two widely used algorithm variants: supernodal (SuperLU: https://portal.nersc.gov/project/sparse/superlu) and multifrontal (STRUMPACK: http://portal.nersc.gov/project/sparse/strumpacK). STRUMPACK is further enhanced with scalable preconditioning using hierarchical matrix algebra. Both libraries are purely algebraic, applicable to many application domains. We will address several Exascale challenges, with the following focus areas: (1) Develop novel approximation algorithms that have lower arithmetic and communication complexity with respect to the size of the input matrix; (2) Develop new parallelization strategies that reduce inter-process communication and expose task parallelism and vectorization for irregular computations involving sparse data structures to better use on-node resources; (3) Integrate our software into higher-level algebraic solvers such as hypre, PETSc, Trilinos, and collaborate with ECP teams for application-specific and hardware-specific tuning of the parameters space to achieve optimal efficiency.

Our solver technology is essential for ECP, because many codes expected to run on Exascale machines need solutions of sparse algebraic systems, and many high-fidelity simulations involve large-scale multiphysics and multiscale modeling problems that generate highly ill-conditioned and indefinite algebraic equations, for which pure iterative methods cannot converge to the solution. The factorization-based algorithms being developed herein represent an important class of methods that are indispensable building blocks for solving those numerically challenging problems. Our software can often be used as a reliable standalone solver, or as a preconditioner for Krylov methods, or as a coarse grid solver in multigrid methods.

Key Challenges At Exascale we need to address several major challenges: decreasing amount of memory per core, increasing impact of communication cost and load imbalance, and increasing architectural heterogeneity. Our new design of algorithms and codes must focus on reducing communication and synchronization and task scheduling instead of floating point operation throughput. In sparse factorization methods, we expect new bottlenecks in parts of the code that previously received little attention. For example, the preprocessing step involves numerical pivoting for selecting stable pivots and symbolic factorization, which do not yet parallelize well on manycore architectures with fine-grained parallelism. At Exascale, direct solvers are more likely to be used in a preconditioning strategy, for example, in block Jacobi preconditioning, in domain decomposition methods or as coarse-grid solvers in algebraic multigrid, which requires repeated triangular solves. The challenge here is to mitigate the low arithmetic intensity and high degree of data dependency.

Compared to iterative methods, the primary bottleneck of direct solvers is the asymptotically higher growth in memory need and floating point operations, especially for problems from three-dimensional geometry. It is imperative to develop new factorization methods that require much less memory and data movement.

Solution Strategy We will address these challenges in several thrust areas. The new techniques will be implemented in the two software packages SuperLU and STRUMPACK. The former is a widely used sparse direct solver based on supernodal factorization and the latter is a newer direct solver/preconditioner package based on multifrontal factorization and hierarchical low-rank matrix structures.

The improvements for SuperLU will be mainly in two areas: (1) develop the communication-avoiding 3D factorization and triangular solve algorithms and codes that have provably lower communication complexity; (2) develop a synchronization-avoiding triangular solve code to enable more overlap of communications of different processes at different substitution steps; (3) develop new multi-GPU codes for both symbolic preprocessing step and numerical factorization and solve steps.

In addition to exploiting structural sparsity as SuperLU does, STRUMPACK also exploits data sparseness in the dense blocks of sparse factors using low-rank representations, which leads to linear scaling $O(n)$ or $O(n \log n)$ memory and arithmetic complexity for PDEs with smooth kernels. The developments for STRUMPACK will focus on several areas: (1) develop robust stopping criteria — both absolute and relative — for adaptive (incremental) randomized sampling schemes to reveal numerical ranks in the low-rank compression routine. The goal is to use enough samples for stability, but not too many for efficiency; (2) add OpenMP support for both HSS compression and ULV factorization routines, especially use OpenMP task construct to support irregular parallelism; (3) reduce MPI communication in all stages of the code, including HSS construction, ULV factorization and triangular solve; (4) in addition to HSS, develop codes to support other simpler low-rank formats, such as HOLDR and BLR. The HSS format has asymptotically lower complexity than HOLDR and BLR, but has a larger prefactor constant. We expect HSS to be more useful for large-scale problems while HOLDR and BLR are more useful for mid-range problems; (5) work with ECP application teams to examine their specific problem characteristics and develop the best clustering/ordering methods to reveal low-rank structures.

Recent Progress We mainly focus on the multi-GPU developments. For SuperLU, the developments are on parallel symbolic factorization and triangular solve with one-sided communication using NVSHMEM. All the experiments are performed on Summit.

For STRUMPACK we improved the GPU off-loading code. We ported the preconditioner based on block low-rank compression to distributed memory systems. We developed an interface from Trilinos to STRUMPACK.

We also worked with the ECP application ExaSGD team, applying our sparse solvers to the linear systems coming from AC optimal power flow problems. The linear systems arising from the Interior Point optimization loops are highly ill-conditioned, and zero-pivots are encountered during numerical factorization. We improved both STRUMPACK and SuperLU to deal with the situation and allow the factorization to succeed and recover the solution accuracy by iterative refinement.

The other algorithmic changes and the results are detailed below.

STRUMPACK

- We added GPU kernels for the extend-add (gather-scatter) operation used in the sparse solver, so that the entire factorization (or a subset fitting in device memory) can be off-loaded without requiring excessive amounts of data movement between host and device. On a single summit node, the new GPU code is up to 8x faster than the old algorithm.
- STRUMPACK now supports AMD GPUs through HIP, and the hipBLAS and rocSOLVER libraries.
- We added a distributed memory block low-rank preconditioner to the sparse solver. For a medium sized 3D 175^3 high frequency Helmholtz problem, the new preconditioner is 3.7x faster than the sparse direct solver, and 2x faster than our previous state of the art preconditioner based on Butterfly compression.

SuperLU

- Finished the first version of the multi-GPU path-based traversal algorithm for parallel symbolic factorization, including supernode detection algorithm. The new code showed strong scaling up to 31x speedup on 44 Summit GPU nodes.
- Implemented the single precision LU factorization with double precision iterative refinement. Initial tests show up to 50-60% speedup on 1 node Summit using 6 CPU cores and 6 GPUs.
- Improved the user interface for the 3D code base: developed the new redistribution routine, so that the users do not need to worry about setting up proper submatrices on the 2D layer of the 3D process grid.

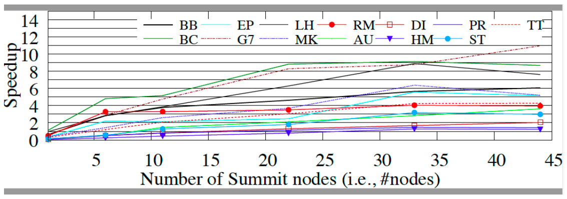


Figure 47: SuperLU symbolic factorization: GPU speedup over CPU, 13 matrices

Next Steps Our future efforts will focus on the following areas: For STRUMPACK, we will work to improve the performance of the triangular solve on the GPU, and add GPU acceleration to the block low-rank solver and preconditioner. For SuperLU, we will work towards release of the GPU-enabled 3D code, including both numerical factorization and triangular solve on multi-GPUs, and release of GPU-enabled symbolic factorization code. Furthermore, we will apply the GPTune autotuner developed from xSDK4ECP project to conduct comprehensive tuning in the parameter space for both STRUMPACK and SuperLU, for the ECP applications and on the pre-exascale machines.

4.3.11 WBS 2.3.3.07 Sub-project: FFTX

Overview The use of Fast Fourier Transforms (FFTs) span a broad range of DOE science applications, including ones represented in the Exascale applications space. In the FFTX project (`https://github.com/spiralgen/fftx`), we are developing a new package for supporting FFT applications on exascale architectures. Our approach is based on the following ideas: a C++ high-level API that can express a more complete set of use cases as a composition of operators including FFTs, but also computation of and multiplication by a symbol, and padded inputs / pruned outputs; a toolchain based on the SPIRAL toolset, an open-source toolchain for FFT developed at CMU, that enables the specialization of the FFT calculation and its surrounding use case calculations (scaling, data layout transformation, marshalling/unmarshalling for communication), using code generation, symbolic analysis, automatic performance tuning, and applications-specific code generation; and a workflow that automates the code generation and integration into the applications software. We are using specific ECP applications and target ECP exascale platforms to provide a focus for this work.

Key Challenges There are several challenges to the FFT-based simulations on exascale systems.

(1) Performance engineering on accelerator-based systems. The traditional approach to FFT calculations is to provide library functions for computing FFTs, with user code to calculate the remaining parts of the algorithm. Such an approach can lead to less than the theoretically maximum performance, specifically due to the larger amounts of data motion due to the coarse data granularity of the library interface. This can be addressed to some extent using a finer-grained interface, but only incompletely, and with code that complicated and difficult to maintain.

(2) Performance portability. High performance is difficult to obtain on a single accelerator-based platform, and the low-level code required to obtain high performance as described in (1) is likely to change as one moves onto different hardware and programming systems.

(3) The need for an open-source ecosystem. One of the major successes of the FFTW software is that it was open source, and available as a starting point for any FFT-based application. However, FFTW has not been supported since the mid-2000's, and is not being ported to GPUs. In the absence of a new open-source FFT code base, ECP will be entirely dependent on what is provided by the vendors, which is a significant risk to the overall success of the ECP applications that depend on FFTs.

Solution Strategies Our approach is based on SPIRAL, an open-source framework for expressing the family of algorithms that includes FFT applications in a high-level notation that can be parsed and represented as nontrivial decompositions of low-level algorithmic fragments (analogous to FFTW codelets). This decomposition can then be transformed into low-level code to be called by the application. The decomposition and code generation is done in a platform-specific fashion for optimal performance, and multiple variants can be generated for the purpose of autotuning. FFTX is building out from SPIRAL in the following ways. First, we are developing a C++ API that expresses high-level components of FFT-based algorithms in a mathematically concise fashion. Executing C++ programs written in terms of this interface generates a SPIRAL symbolic representation of these algorithms, which is then transformed by SPIRAL into optimized low-level code. The class of algorithms expressible in the FFTX API (and in the SPIRAL symbolic representation) includes many of the ECP algorithms, such as convolutions of various types, and plane-wave calculations for density functional theory calculations. Second, we are developing novel algorithmic analysis and code generation techniques specific to the requirements of exascale systems, in particular to reduce data footprint and data movement in ways that are impractical, if not impossible, for a human programmer to do. Finally, we are developing an automated workflow for managing the process that goes from an FFTX specification to code generation to compiling and linking to a user's application.

This approach addresses the challenges given above. The "whole-algorithm" approach to expressing and generating optimized code for FFT-based algorithms enables optimizations that are not feasible in the classical library approach; for example, fine-grained interleaving of sub-steps in a multidimensional FFT and parts of the symbol calculation in a convolution, or eliminating unused calculations for padded inputs or pruned outputs. The high-level specification of the algorithms through the FFTX API does not change when moving between platforms, this providing performance portability from the application developer's standpoint. Also, other parts of the toolchain are reused across platforms, such as the higher-level stages of the SPIRAL representation. Finally, the entire SPIRAL and FFTX software stack is open source. SPIRAL is publicly available, as will be various FFTX capabilities as they are completed.

Recent Progress We have a complete end-to-end FFTX implementation of the PSATD algorithm for solving Maxwell's equations on CPUs, a key component of the ECP WarpX application. This includes: a FFTX API representation of the PSATD algorithm; SPIRAL code generation of c code; and linking to a version of the WarpX ECP application that uses PSATD, including coupling of FFTX and AMReX data representations. We have performed side-by-side comparisons of the results obtained by the standard FFTW-based WarpX implementation, and that obtained from FFTX, and they agree to within roundoff error. We have also developed strategies for obtaining high-performance on multi-gpu and multi-node subsets of summit.

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

- Deliver high performance on Summit for ECP AD use cases, including plane-wave, PSATD, and periodic convolutions, both on single-GPU and multi-GPU configurations.
- Code generation for Tulip / Frontier platform, based on the HIP programming tools.
- Complete release version, documentation for release.

4.3.12 WBS 2.3.3.12 Sub-project: SUNDIALS

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

The SUNDIALS suite of packages [132] provides efficient and adaptive time integrators and nonlinear solvers. The packages are written using encapsulation of both data structures and solvers, thus allowing easy incorporation into existing codes and flexibility to take advantage of new solver technologies and packages. SUNDIALS provides both multistep and multistage methods designed to evolve stiff or nonstiff ordinary (ODE) and differential algebraic (DAE) systems with efficient accuracy-driven time step selection. SUNDIALS also provides both Newton and fixed point (with optional acceleration) nonlinear solvers and scaled Krylov methods with hooks for user-supplied preconditioners. Users can also supply their own nonlinear and linear solvers under the integrators. SUNDIALS is released with data structures supporting several programming environments. Users can employ these supplied structures or provide their own.

Through software infrastructure developments, this project is enabling the efficient and robust SUNDIALS time integrator packages to easily interoperate with applications evolving time dependent systems as well as with external linear and nonlinear solver packages developed for exascale computing. In addition, this project is providing support for integrating several independent ordinary differential equation systems simultaneously on GPUs as part of multiphysics applications. Lastly, this project is supporting the deployment and use of SUNDIALS packages within ECP applications, mainly through incorporation into the discretization-based Co-Design Centers, AMReX and CEED.

Key Challenges Current implementations of efficient time integrators face challenges on many fronts. First, applications need both efficient integrators and ones that can interface easily with efficient linear algebra packages to solve subservient linear systems. In addition, integrators and their interfaces to both solver libraries and applications must be frequently updated to keep up with rapid advances in system architectures. Some ECP applications require the solution of many small systems of ODEs in parallel on GPUs giving rise to the need for a GPU-enabled ODE integrator that can be used in parallel for many systems at once and be able to run on multiple GPU-based architectures with differing programming models. Lastly, ECP applications require assistance incorporating new linear solvers underneath the integrators and in updating their interfaces to optimally use integrators on new platforms.

Solution Strategy This project includes a number of implementation activities that will prepare the SUNDIALS suite of time integrators for systems found in ECP applications. A major activity is developing support for evolving multiple systems of ODEs in parallel on AMD GPUs with the CVODE multistep ODE integration package. To meet this need, the SUNDIALS team has added support for assigning data structures and solvers to a particular GPU stream, thus making it possible for multiple instances of CVODE to simultaneously utilize the GPU in parallel. CVODE has also been equipped with interfaces to a batched direct linear solver capable of using an NVIDIA GPU. Currently, new interfaces are being developed to provide these capabilities on AMD GPUs, as are expected for Frontier. While these interfaces are straightforward for the vector operations, linear solvers that are AMD-GPU capable are not yet generally available. SUNDIALS is working with ECP linear solver packages, such as Gingko and MAGMA, to take advantage of HIP-based solvers that they develop and that will be efficient for the expected systems.

In addition, this project is working to evaluate and optimize integrator performance within its ECP user applications. A small test suite allowing easier evaluation of performance on new platforms is being developed. This year, SUNDIALS will stand up this new test suite within the GitLab CI for testing from OLCF systems with the goal of using this infrastructure to evaluate performance. Moreover, the SUNDIALS team is adding a performance assessment layer and enabling use of the ECP ST Caliper package for performance testing underneath that layer. The SUNDIALS team plans to work with the AMPE phase field code (ExaAM project), PELEC, PELELM, and Nyx users to apply these tools in assessing and optimizing SUNDIALS' performance within their applications.

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

Recent Progress SUNDIALS had four releases this past year, including new features in direct support of ECP application needs. In particular, releases in March and May 2020 included a new matrix implementation that interfaces to the sparse matrix implementation from the NVIDIA cuSparse library, new specialized fused CUDA kernels in CVODE which offer better performance on smaller problems when using CVODE with CUDA, new routines that support the ability to control kernel launch parameters for the CUDA vector and matrix modules, and new diagnostic routines that support load balancing efforts by making information on the difficulty of solves more accessible to users. These features directly support capabilities needed in solving many small ODE systems simultaneously and have been integrated into the SUNDIALS use from AMReX-based applications, including Nyx, PELEC, and PELELM.

In addition, the May 2020 release also included new code to support integration of an ODE system while projecting onto a constraint manifold. This capability, previously in a one-off package, CPODES, is needed by the AMPE phase field code in the ExaAM project. The SUNDIALS team has worked with the AMPE team to incorporate the new version of CVODE into their software stack.

Lastly, the Sept. 2020 release included a new feature in the ARKODE package to support integration of systems with a time-dependent mass matrix. This feature is needed by the MFEM high order discretization package in the CEED Co-Design center.

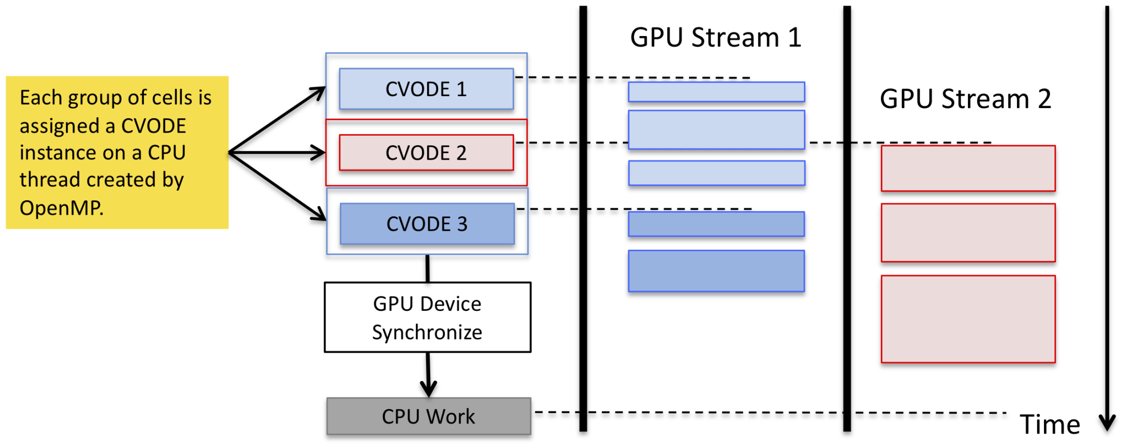

Each group of cells is assigned a CVODE instance on a CPU thread created by OpenMP.
CVODE 1
CVODE 2
CVODE 3
GPU Device Synchronize
CPU Work
GPU Stream 1
GPU Stream 2
Time

Figure 48: Illustration of SUNDIALS' hybrid, OpenMP + GPU approach to integrating the many small ODE systems that arise in the PELE and Nyx applications. In this example, three distinct groups, formed by grouping the independent ODE systems arising in AMR grid cells, of ODE systems are integrated with CVODE. The groups, each defining a larger ODE system, are distributed across CPU threads with OpenMP. On each thread, a distinct and independent CVODE instance solves the larger ODE system. CVODE launches GPU kernels in streams, allowing some threads to operate simultaneously.

Next Steps During the remainder of FY21, this project team will:

1. Release SUNDIALS with vector and solver support for AMD GPUs.
2. Document performance of SUNDIALS within two ECP applications.
3. Expand SUNDIALS support for Intel GPUs.
4. Develop a performance test suite and document performance of SUNDIALS using GitLab and Caliper.
5. Continue to support AMReX and CEED Co-Design Centers in their use of SUNDIALS.

4.3.13 WBS 2.3.3.12 Sub-project: hypre

Overview The *hypre* software library [133, 134] provides high performance preconditioners and solvers for the solution of large sparse linear systems on massively parallel computers, with particular focus on algebraic multigrid solvers. One of *hypre*'s unique features is the provision of a (semi)-structured interface, in addition to a traditional linear-algebra based interface. The semi-structured interface is appropriate for applications whose grids are mostly structured, but with some unstructured features. Examples include block-structured grids, composite grids in structured adaptive mesh refinement (AMR) applications, and overset grids. These interfaces give application users a more natural means for describing their linear systems, and provide access to methods such as structured multigrid solvers, which can take advantage of the additional information beyond just the matrix. Since current architecture trends are favoring regular compute patterns to achieve high performance, the ability to express structure has become much more important. The *hypre* library provides both unstructured and structured multigrid solvers, which have shown excellent scalability on a variety of high performance computers, e.g Blue Gene systems (unstructured solver BoomerAMG has scaled up to 1.25 million MPI cores with a total of 4.5 million hardware threads). It is used by many ECP application teams, including ExaAM, Subsurface, ExaWind, CEED, and more. It requires a C compiler and an MPI implementation, but it also runs in an OpenMP environment. It also has GPU capabilities.

Key Challenges While *hypre*'s solvers contain much parallelism, their main focus is the solution of sparse linear systems, leading to very large demands on memory bandwidth. In addition, the use of multiple levels, while greatly aiding convergence of the solvers, leads to decreasing systems sizes, number of operations and parallel efficiencies on coarser levels. Particularly the unstructured algebraic multigrid solver BoomerAMG[135], which is *hypre*'s most often used preconditioner, suffers from increasing communication complexities on coarser levels. Coarse grid operators are generated by multiplying three matrices leading to increasing numbers of nonzeroes per row in the resulting matrices and with it increasing numbers of neighbor processes. While BoomerAMG's solve phase mainly consists of matrix vector products and smoothing operations, which are fairly straight forward to parallelize, even on a GPU, its setup phase is highly complex, including many branches, a lot of integer operations as well as some sequential passages. Current interpolation strategies that lead to best convergence and performance on distributed memory machines are not suitable for implementation on GPUs or similar architectures requiring extreme parallelism. Since *hypre* is a mature product with many solvers and interdependent features, any significant changes that affect the whole library, are tedious and require much testing to ensure that the library stays backward compatible and no features are broken.

Solution Strategy Since computer architectures continue to change rapidly, it was important to come up with strategies that will facilitate future porting of the software. Therefore we developed and implemented a new memory model that addresses the use of different memory locations. Since the upcoming computer architectures are heterogeneous with accelerators, we focus on enabling *hypre* for GPUs. We have looked into various options, such as the use of CUDA, OpenMP 4.5, as well as RAJA and Kokkos. We limited the latter three options to the structured interface and solvers which are more natural candidates for such an approach due to their use of macros, called BoxLoops, for loops. We adopted a modular approach for the unstructured interface, which relies on the restructuring the solver components to use smaller kernels that are and/or will be implemented in CUDA for Nvidia GPUs. We will investigate the use of vendor conversion tools from CUDA to HIP and SYCL to port the unstructured solvers to upcoming exascale computers.

Recent Progress Previously, we enabled the structured solvers, SMG and PFMG[136], both setup and solve phase, to completely run on GPUs, using both CUDA or OpenMP4.5, or use optional RAJA and Kokkos. For our unstructured AMG solver BoomerAMG, we had implemented suitable CUDA kernels for setup and solve phase, which allowed AMG to run on GPUs for specific settings, but did not include our best interpolation operators. Recently, we added CUDA capabilities to create and assemble IJ matrices and vectors. Since our best interpolation operators are not suitable for GPU implementation, we designed a new class of interpolation operators based on sparse matrix operations[137] and implemented it on GPUs. We also ported aggressive coarsening to the GPU, which leads to decreased memory complexities and can also reduce overall run times. This included the implementation of a second strength matrix, required to get even coarser grids, and several two-stage interpolation operators also based on matrix-matrix operations and capable to deal with grid points that are further apart. Figure 49 and Figure 50 show two weak scaling studies comparing GPU and CPU implementations of AMG-PCG on Lassen, using 4 MPI tasks per node with 1 GPU per MPI task for the GPU version and 10 OpenMP threads per MPI task for the CPU version.

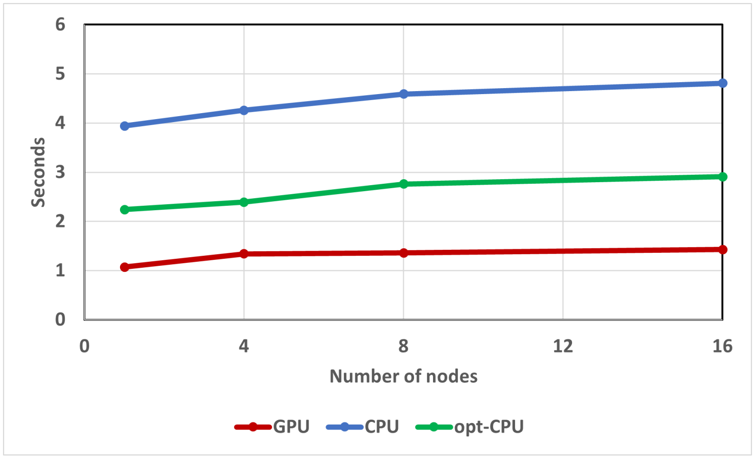


Figure 49: Weak scaling study for AMG-PCG applied to a 3D 27pt diffusion problem on Lassen with 8M grid points per node comparing total run times (setup and solve) on GPUs with the new mm-ext+i interpolation (GPU), and on CPUs using ext+i interpolation (CPU), and adding aggressive coarsening with multipass interpolation on the first level (opt-CPU)

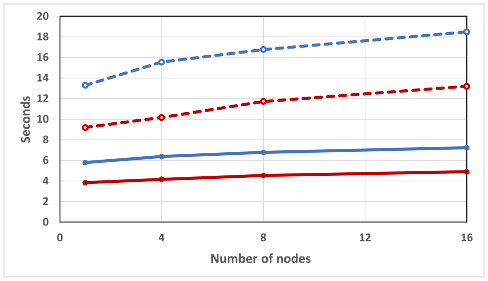


Figure 50: Weak scaling study for AMG-PCG applied to a system of coupled Poisson problems with 3 variables per grid point on Lassen with 8M grid points (24M dofs) per node comparing CPU (dashed) and GPU (solid) total run times with (red) and without (blue) aggressive coarsening on the first level. Here CPU and GPU runs use mm-ext+e interpolation and two-stage mm-ext+e interpolation when using aggressive coarsening.

Next Steps We will pursue the following tasks:

- We will continue to add new GPU capabilities to *hypre* and improve the performance of current capabilities. We will thoroughly investigate the performance on Nvidia GPUs and begin porting to AMD GPUs.
- We also investigate and improve the performance of other unstructured solvers in *hypre*, such as AMS and ILU and port components to GPUs where needed.

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

4.3.14 WBS 2.3.3.13 CLOVER

Mathematical libraries are powerful tools to make better use of Exascale architectural features and are central for application projects to efficiently exploit the available computing power. The high-level objective of CLOVER is to provide scalable, portable numerical algorithms that facilitate efficient application simulations on Exascale computers. With the intention of generating synergies by facilitating vivid cooperation among the distinct project focus efforts and expert knowledge transfer, CLOVER was designed as a merger of the SLATE, FFT-ECP, PEEKS, and Kokkos Kernels projects, each being complementary in focus but similar in the need for hardware-specific algorithm design expertise: SLATE focuses on Exascale-capable dense linear algebra functionality; FFT-ECP's scope is providing robust and fast calculation for 2D and 3D FFT routines; PEEKS delivers production-ready, latency-tolerant and scalable preconditioned iterative solvers; Kokkos Kernels delivers performance-portable kernels for on-node sparse and dense linear algebra and graph algorithms. Together, these projects form a robust ecosystem of numerical base functionality for Exascale computers.

4.3.15 WBS 2.3.3.13 *CLOVER Sub-project FFT-ECP*

Overview The FFT-ECP project provides sustainable high-performance multidimensional Fast Fourier Transforms (FFTs) for Exascale platforms through the *Highly Efficient FFTs for Exascale* (**heFFTe**) library [138]. HeFFTe leverages established but *ad hoc* software tools that have traditionally been part of application codes, but not extracted as independent, supported libraries.

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

- Collect existing FFT capabilities from ECP application teams;
- Assess gaps, extend, and make available various FFT capabilities as a sustainable math library;
- Explore opportunities to build multidimensional FFTs while leveraging on-node concurrency from batched FFT formulations;
- Focus on capabilities for Exascale platforms.

FFTs are used in many applications including molecular dynamics, spectrum estimation, fast convolution and correlation, signal modulation and many wireless multimedia applications. The distributed 3D FFT is one of the most important routines used in molecular dynamics (MD) computations, and its performance can affect MD scalability. The performance of the first principles calculations strongly depends on the performance of the FFT solver that performs many FFTs of size $\approx 10^7$ points in a calculation that we call batched FFT. Moreover, Poisson PDE-type equations arising from many engineering areas, such as plasma simulation and density fields, need to solve FFTs of size larger than 10^9. More than a dozen ECP applications use FFT in their codes. ECP applications that require FFT-based solvers suffer from the lack of fast and scalable 3D FFT routines for distributed-heterogeneous parallel systems as the ones projected for the upcoming exascale computing systems. To address these needs, heFFTe functionalities are first delivered to CoPA projects using LAMMPS (molecular dynamics) and HACC (Hardware Accelerated Cosmology Code).

The heFFTe software stack is illustrated in the left-hand side of Figure 51, while the main components of the heFFTe framework are illustrated in the right-hand side of Figure 51. The first and last step address the need for a flexible FFT API to take application-specific input and output (bricks/pencils), including arbitrary initial decompositions.

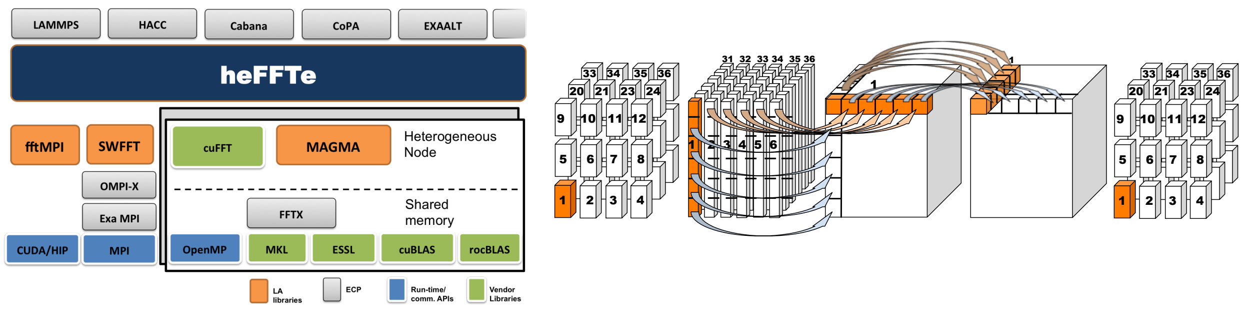


Figure 51: **Left**: the heFFTe software stack. **Right**: 3D FFT computational pipeline in heFFTe with: 1) Flexible API for application-specific input and output, including bricks/pencils/etc.; 2) Efficient packing/unpacking and MPI communication routines; 3) Efficient 1D/2D/3D FFTs on the node.

Key Challenges

1. **Communication costs:** Communication costs are main bottleneck on current systems; this includes low node bandwidth (relative to high compute capabilities), sub-optimal accelerator-aware MPI communications, and encountered performance degradations in MPI implementations.
2. **Application specifics:** ECP applications that require FFT-based solvers suffer from the lack of fast and scalable FFTs for distributed-heterogeneous parallel systems as the ones projected for the upcoming

exascale computing systems. Also, ECP applications need different application-specific versions of FFTs, and dictate parallelism and data distributions (where is the data, how is distributed, what is the parallelism, etc.). This requires application knowledge and API designs with a suitable modular high-performance implementation that is flexible and easy to use and integrate in ECP applications.

3. **Performance portability:** Performance portability across different architectures is always a challenge. This is further exacerbated due to the many application and hardware-specific FFT versions needed.

Solution Strategy

1. **Communications and GPU optimizations:** FFTs are communication bound and a main focus in heFFTe is on algorithmic design to minimize communication and efficient GPU implementations [139, 140]. Other strategies include the use of mixed-precision calculations [141, 142] and data compression for reduced communications (including lossy, e.g., using ZFP compression).

2. **Evolving design:** heFFTe is designed to support the fftMPI and SWFFT functionalities, which are already integrated in ECP applications. Thus, heFFTe benefits directly these applications and provides integrated solutions. More functionalities and application-specific optimizations will be added to support various ECP applications.

3. **Autotuning:** Performance portability will be addressed through use of standards (like 1D FFTs from vendors), portable linear algebra (LA) using MAGMA [143], and parameterized versions that will be tuned across architectures. We have extensive expertise and well proven track record in the development and use of autotuning techniques for important LA kernels [144, 145].

Recent Progress The FFT-ECP team completed two main milestones involving software releases adding numerous stability, performance, and scalability enhancements, as well as new functionalities. HeFFTE 0.2 was released in January 2020 [146], and heFFTe 2.0 was released in September 2020. HeFFTe 2.0 added support for AMD GPUs and bindings for C, Fortran, and Python. HeFFTe now can be installed through *spack* and is compatible with the xSDK community policies, and will be part of the next xSDK release. HeFFTe is also integrated in CoPA projects using LAMMPS and HACC. HeFFTe 2.0 demonstrates very good strong scalability and performance that is close to 90% of the roofline peak [147]. (see Figure 52).

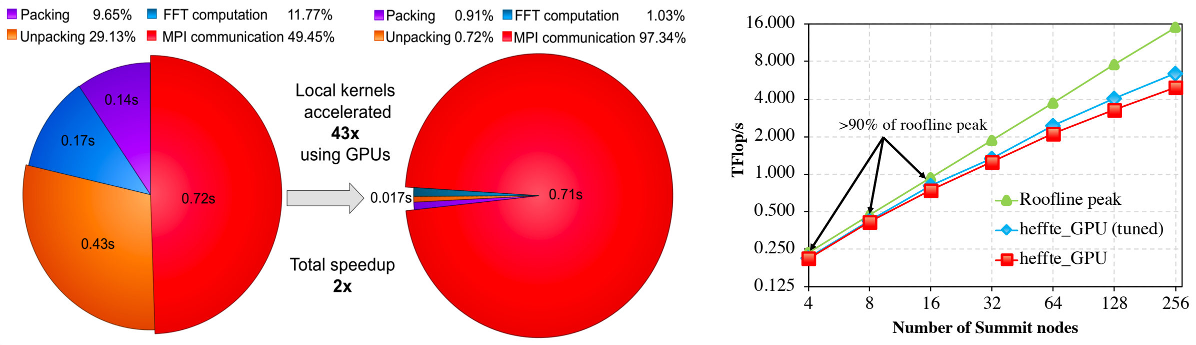


Figure 52: Left: heFFTe acceleration of 1024^3 FFT on 4 Summit nodes. Note: nodal computations are accelerated 43×. **Right**: heFFTe strong scalability on 1024^3 FFT on up to 256 nodes (×6 V100 GPUs; double complex arithmetic; starting and ending with bricks; performance assumes $5N^3 log_2 N^3$ flops).

Next Steps Next steps of work are adding multidimensional FFTs and optimizations for real data. This will include the development of R2C and C2R DFTs and their integration and specific optimizations in ECP applications. Further integration and use will be added to CoPA applications and the ExaAM project. Optimizations will be done for AMD GPUs and support added for Intel GPUs.

4.3.16 WBS 2.3.3.13 CLOVER Sub-project Kokkos Kernels

Overview The Kokkos Kernels [2] subproject primarily focuses on performance portable kernels for sparse/dense linear algebra, graphs, and machine learning, with emphasis on kernels that are key to the performance of several ECP applications. We work closely with ECP applications to identify the performance-critical kernels and develop portable algorithms for these kernels. The primary focus of this subproject is to support ECP application needs, develop new kernels needed with an emphasis towards software releases, tutorials, boot camps and user support. The Kokkos Kernels project also works closely with several vendors (AMD, ARM, Cray, Intel, and NVIDIA) as part of both the ECP collaborations and NNSA's Center for Excellence efforts. These collaborations will enable vendor solutions that are targeted towards ECP application needs.

Key Challenges There are several challenges in allowing ECP applications move to the hardware architectures announced in the next few years. We highlight the four primary challenges here:

1. The next three supercomputers that will be deployed will have three different accelerators from AMD, Intel and NVIDIA. While we have been expecting diversity of architectures, three different architectures in such a short timeframe adds pressure on the portability solutions such as Kokkos Kernels to optimize and support the kernels on all the platforms.
2. The design of several ECP applications and a software stack that rely on a component-based approach results in an extremely high number of kernel launches on the accelerators, which results in the latency costs becoming the primary bottleneck in scaling the applications.
3. The change in the needs of applications from device-level kernels to smaller team-level kernels. Vendor libraries are not ready for such a drastic change in software design.
4. The reliance of ECP applications on certain kernels that do not port well to the accelerator architectures.

These challenges require a collaborative effort to explore new algorithmic choices, working with the vendors to incorporate ECP needs into their library plans, to develop portable kernels from scratch, and to deploy them in a robust software ecosystem. The Kokkos Kernels project will pursue all of these choices in an effort to address these challenges.

Solution Strategy Our primary solution strategy to address these challenges are:

1. **Codesign portable kernels with vendors and applications:** We rely on codesign of Kokkos Kernels implementations for specializations that are key to the performance of ECP applications. This requires tuning kernels even up to the problem sizes that are of interest to our users. Once we have developed a version, we provide these to all the vendors so their teams can optimize these kernels even further in vendor-provided math libraries.
2. **Emphasis on software support and usability:** The Kokkos Kernels project devotes a considerable amount of time working with ECP applications, integrating the kernels into application codes, tuning for application needs, and providing tutorials and user support. We invest in delivering a robust software ecosystem that serves the needs of diverse ECP applications on all platforms of interest.
3. **Invest in algorithmic research to reduce latency costs and new accelerator focused approaches:** To resolve latency cost issues, the Kokkos Kernels team is considering several solutions from computer science perspectives and also from algorithmic applied mathematics perspectives. For example, from a computer science perspective, we are focusing on the use of streams or other latency reducing techniques such as cuda graphs. From the applied mathematics perspective we are developing new algorithms such as cluster-based approaches for preconditioners, such as Gauss-Seidel precondtioners, to reduce the number of kernel launches.

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

Recent Progress

1. Kokkos Kernels has developed team-level linear algebra kernels for several BLAS and LAPACK operations so that ECP applications can use these foundational operations within their device level code. This key design championed by the Kokkos Kernels team is becoming more common place with several vendors adopting such a design. This design reduces synchronization overhead and encourages reuse of the data in the memory hierarchy between several BLAS or LAPACK operations with team level synchronization. This has resulted in better performance in applications like SPARC CFD simulation.
2. Kokkos Kernels preconditioners such as Symmetric Gauss Seidel precondtioners are integrated into the Exawind application. The multicoloring based Symmetric Gauss Seidel precondtioner has resulted in up to 4.5x improvement in overall solve time of the SGS solver.
3. Kokkos Kernels BLAS and sparse linear algebra kernels were integrated into the spectral partitioner of the Exagraph project. Using Kokkos Kernels results in faster spectral partitioning than vendor provided implementations. This was the result of careful tuning of the kernels for Exagraph needs.
4. Kokkos Kernels team has developed and integrated tutorial materials to the Kokkos tutorials. The tutorials are maintained as a common resource for the entire Kokkos ecosystem.

Next Steps Kokkos Kernels team is focused on:

1. **A major software release**: Kokkos ecosystem 3.0 release will be available to the ECP applications in FY 2020. This includes several new kernels that are requested by ECP applications, performance improvements of kernels that are already being used by ECP applications, support for new architectures such as ARM based systems, and several software changes such as support of standalone CMake.
2. **Developing new kernels to reduce synchronization costs**: Kokkos Kernels team is working on kernels that reduce the number of kernels launches by focusing on block-based approaches. This will allow further performance in ECP applications such as Exawind and EMPIRE.
3. **Collaboration with vendors**: Kokkos Kernels team is working with vendor libraries team to incorporate ECP application needs in the vendor library roadmap. Several changes from vendors are expected in FY20. These changes will be added to Kokkos Kernels so ECP applications can get access to the improvements.

4.3.17 WBS 2.3.3.13 CLOVER Sub-project PEEKS

Overview The PEEKS subproject is a focused team effort to advance the capabilities of the ECP software stack in terms of communication-avoiding Krylov solvers and advanced preconditioning techniques featuring fine-grained parallelism. Previously developed techniques that are available as prototype codes – as well as novel algorithm developments – are turned into production-quality implementations and integrated into the ECP software ecosystem as part of the Trilinos [3] and the Ginkgo [4] software stacks.

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

1. Fine-grained parallelism in a single node that has to be exploited efficiently by the iterative solver and the preconditioner.
2. Rising communication and synchronization cost as the computational power is growing much faster than memory power, resulting in increased pressure on the bandwidth of all cache/memory levels.

All challenges require the redesign of existing iterative solvers with respect to higher parallelism, a reduced number of communication and synchronization points, favoring computations over communication, and adopting multiprecision algorithms for efficient hardware utilization.

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

Solution Strategy The primary thrusts of the PEEKS project are:

1. **Architecture-portable software design:** In the Ginkgo C++ software [148], we design and develop a next-generation sparse linear algebra library able to run on multi- and manycore architectures. The library design decouples algorithm implementations from hardware-specific kernel implementations, thereby acknowledging the importance of platform portability and allowing for extensibility as well as architecture-specific kernel optimization.
2. **Sustainability efforts:** The Ginkgo and Trilinos software development adheres the Better Scientific Software (BSSw) design principles [149] that ensure production-quality code by featuring unit testing, automated configuration and installation, Doxygen code documentation, as well as a continuous integration and continuous benchmarking framework [150]. Ginkgo and Trilinos are open source effort licensed under BSD 3-clause and included in the xSDK and I4S software packages.
3. **Pipelined and CA Krylov methods:** We realize pipelined and communication-avoiding Krylov methods in production-quality code, and we are actively collaborating with the ECP ExaWind project to integrate our new features into their application [151].
4. **Memory Precision Decoupling:** In collaboration with the ECP xSDK multiprecision effort, we are working on sparse linear algebra iterative methods and preconditioners that reduce runtime by compressing data before invoking memory operations, such as the adaptive precision block-Jacobi preconditioner [152] and the compressed basis GMRES [153].

Recent Progress

1. The Ginkgo library realized full native support for NVIDIA GPUs (via CUDA) and AMD GPUs (via HIP) and became a role model for platform portability [154].
2. The production-ready implementation of the first parallel threshold ILU preconditioner (ParILUT [155]) compensates via algorithmic improvement for 5 years of hardware development (see Figure 53).
3. We implemented and released five variations of communication-avoiding and pipelined Krylov solvers in the Belos Trilinos package.
4. We demonstrated the efficient use of communication-avoiding Krylov methods in Trilinos inside wind turbine simulations of the ECP ExaWind project [151].
5. We developed an initial implementation of a new polynomial preconditioner based on the GMRES polynomial [156].

Next Steps Our next efforts are:

1. **Block-versions of the parallel Incomplete factorization preconditioner:** To better reflect the properties of the ECP application projects, we will deploy blocked versions of the ParILU and ParILUT parallel ILU and parallel threshold ILU preconditioners in the Ginkgo software library.
2. **Intel GPU backend:** We are currently designing a Ginkgo backend for Intel GPU architectures based on the SYCL language.
3. **Compressed Basis Krylov Solvers:** Using the memory accessor designed in the xSDK multiprecision project, we will develop Krylov solvers that reduce the execution time by compressing the Krylov vectors before invoking memory operations.
4. **Problem-specific preconditioners for MFEM:** In collaboration with the ECP CEED cluster, we will design problem-specific preconditioners for matrix-free finite element simulations.
5. **Low-synchronous orthogonalization:** The success of communication-avoiding Krylov methods motivates to push the synchronization limits further by deploying low-synchronous orthogonalization methods. (Collaboration with the ExaWind team at NREL.)

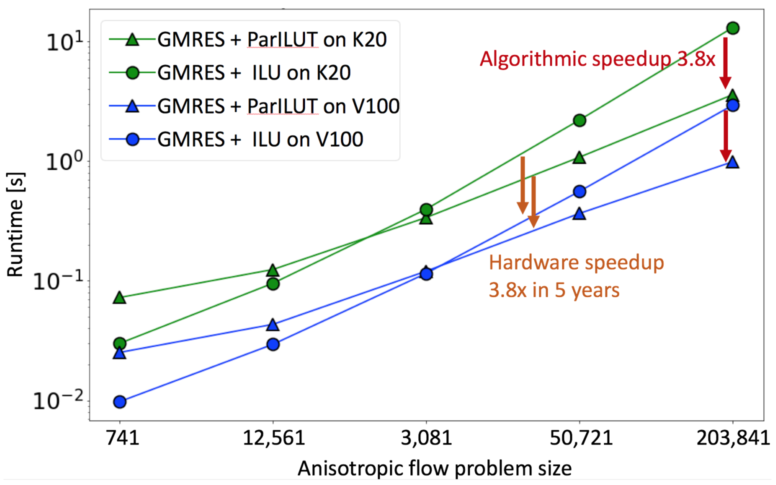


Figure 53: Time-to-solution performance of anisotropic flow problems of different sizes on different hardware architectures: standard ILU(0) vs. new ParILUT.

6. **Parallel incomplete factorization preconditioner application:** With the advances in the parallel incomplete factorization preconditioner generation, the focus increasingly turns to the efficient preconditioner application. We enhance the concept of sparse approximate inverse approximation for incomplete factorization preconditioners, and extend the scope to novel hardware architectures featuring attractive performance in the low-precision regimes.

7. **Polynomial preconditioners:** Polynomial preconditioning is an old idea, but has had limited popularity since it is hard to find good polynomials in the general case. We believe using the GMRES polynomial addresses this issue. Polynomials are also cheap to apply and save communication by reducing the number of inner products. We plan to implement a version using Kokkos that runs on GPU and exascale systems.

4.3.18 WBS 2.3.3.13 CLOVER Sub-project SLATE

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

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

Primarily, SLATE aims to extract the full performance potential and maximum scalability from modern HPC machines with large numbers nodes, large numbers of cores per node, and multiple GPUs per node. For typical dense linear algebra workloads, this means getting close to the theoretical roofline peak performance and scaling to the full size of the machine. This is accomplished in a portable manner by relying on standards such as MPI and OpenMP. Figure 54 shows the role of SLATE in the ECP software stack.

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

Key Challenges

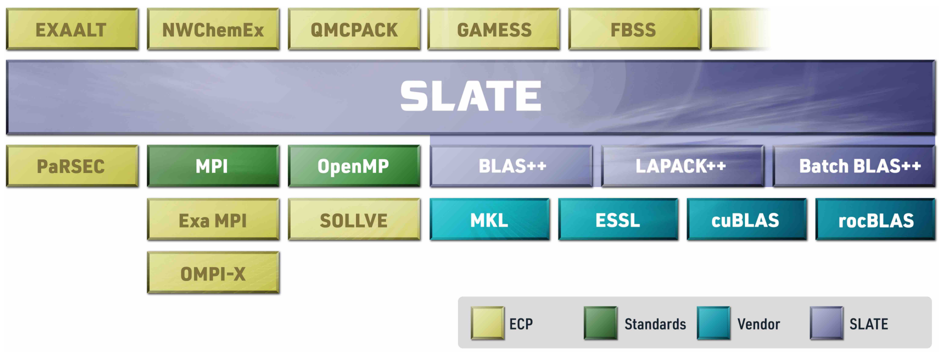


Figure 54: SLATE in the ECP software stack.

1. **Designing from the ground up:** The SLATE project's primary challenge stems from the need to design the package from the ground up, as no existing software package offers a viable path forward for efficient support of GPUs in a distributed-memory environment.

2. **Facing harsh hardware realities:** SLATE is being developed in a difficult hardware environment, where virtually all the processing power is on the GPU side. Achieving efficiency requires aggressive offload to GPU accelerators and careful optimization of multiple bottlenecks, including interconnect technology lagging behind the computing capabilities of the GPUs.

3. **Facing harsh software realities:** SLATE is being developed using cutting-edge software technologies, and relies on modern C++ features and recent extensions to the OpenMP standard, many of which are not fully supported by compilers and their runtime environments. In terms of GPU acceleration, standardized solutions are still in flux.

Solution Strategy

1. **Evolving design:** Due to the inherent challenges of designing a software package from the ground up, the SLATE project started with a careful analysis of the existing and emerging implementation technologies [157], and followed with a phase of laying out the initial design [158]. Since then, the team has rolled out new computational routines and performance improvements quarterly. While we continue to refactor as needed to achieve high performance, the basic design has solidified and been published [159].

2. **Focus on GPUs:** Efficient GPU acceleration is the primary focus of performance engineering efforts in SLATE. Where applicable, highly optimized vendor implementations of GPU operations are used, such as the batched `gemm` routine. Where necessary, custom GPU kernels are developed, as in the case of computing matrix norms. Care is taken to hide communication by overlapping it with GPU computations.

3. **Community engagement:** The SLATE team interacts on a regular basis with the OpenMP community, represented in ECP by the SOLLVE project, and with the MPI community, represented in ECP by the OMPI-X project and the Exascale MPI project. The SLATE team also engages the vendor community through our contacts at Cray, IBM, Intel, NVIDIA, AMD, and ARM.

Recent Progress During 2020, the SLATE team expanded the Hermitian eigenvalue solver to the generalized Hermitian problem of the forms $Ax = \lambda Bx$, $ABx = \lambda x$, or $BAx = \lambda x$. These are of strong interest in mechanics and chemistry applications, among others. We also implemented the polar decomposition with the QDWH algorithm, wrote a Users' Guide and Developers' Guide to document the public and internal APIs, and added native C and Fortran 2003 APIs that access SLATE's native matrix types from outside C++. Further work was done on performance enhancements, with notable gains for BLAS (gemm, herk), norms, Cholesky and QR factorizations. A new build system using CMake and Spack has been developed, in collaboration with the NWChemEx project to help include BLAS++ and LAPACK++ in their project. All developments are documented in SLATE Working Notes [5]

Next Steps

1. **Port to AMD and Intel platforms:** Originally, SLATE was developed using NVIDIA CUDA and cuBLAS. We are now abstracting the backend to run on AMD and Intel platforms. BLAS++ will serve as a portability layer, with calls to NVIDIA cuBLAS, AMD rocBLAS, or Intel oneMKL, as appropriate for the platform. CUDA kernels will be ported to a combination of HIP, SYCL, and OpenMP offload.
2. **Optimizing QR, eigenvalue, and singular value routines:** These are fundamental routines used by many projects. We have observed these routines that are not performing as well as expected. In some cases, such as eigenvalues, we have already identified improvements to be made to the algorithm, such as refactoring loops to improve parallelism. In other cases, we will analyze traces to identify and correct problems.
3. **Implementing divide-and-conquer algorithm:** For singular value and Hermitian eigenvalue problems, the divide-and-conquer algorithm exhibits better performance and parallel scalability than the traditional QR-iteration based algorithm currently used in SLATE.
4. **Non-symmetric eigenvalue problem:** Computing the non-symmetric eigenvalue problem is significantly more computationally expensive than the Hermitian eigenvalue problem. Our implementation will leverage the latest advances, such as aggressive early deflation, to achieve high performance.

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

4.3.19 WBS 2.3.3.14 ALExa

Overview The ALExa project (*Accelerated Libraries for Exascale*) focuses on preparing the ArborX, DTK, Tasmanian, and ForTrilinos libraries for exascale platforms and integrating these libraries into ECP applications. These libraries deliver capabilities identified as needs of ECP applications: (1) the ability the perform performance portable spatial searches between arbitrary sets of distributed geometric objects (ArborX); (2) the ability to transfer computed solutions between grids with differing layouts on parallel accelerated architectures, enabling multiphysics projects to seamlessly combine results from different computational grids to perform their required simulations (DTK); and (3) the ability to construct fast and memory efficient surrogates to large-scale engineering models with multiple inputs and many outputs, enabling uncertainty quantification (both forward and inverse) as well as optimization and efficient multi-physics simulations in projects such as ExaStar (Tasmanian); and (4) the ability to automatically interface Fortran-based codes to existing large and complex C/C++ software libraries, such as Trilinos advanced solvers that can utilize next-generation platforms.

These capabilities are being developed through ongoing interactions with our ECP application project collaborators to ensure they will satisfy requirements of these customers. The libraries in turn take advantage of other ECP/SW capabilities currently in development, including Trilinos, Kokkos, and SLATE. The final outcome of the ECP project will be a set of libraries deployed to facilities and also made broadly available as part of the xSDK4ECP project.

ArborX

Purpose: ArborX is an open-source library designed to provide performance portable algorithms for geometric search.

Significance: General geometric search capabilities are needed in a wide variety of applications, including the generation of neighbor lists in particle-based applications (e.g., molecular dynamics or general N-body dynamics simulations), density-based clustering analysis (e.g., halo finding or DBSCAN in cosmology) and mesh-mesh interactions such as contact in computational mechanics and solution transfer in multiphysics simulations.

Performance portable search capabilities: Shared memory and GPU implementations of spatial tree construction; shared memory and GPU implementations of various spatial tree queries; MPI front-end for coordinating distributed spatial searches between sets of geometric objects with different decompositions; communication plan generation based on spatial search results; density-based clustering algorithms (DBSCAN).

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

DTK (Data Transfer Kit)

Purpose: Transfers computed solutions between grids with differing layouts on parallel accelerated architectures.

Significance: Coupled applications frequently have different grids with different parallel distributions; DTK is able to transfer solution values between these grids efficiently and accurately.

Mesh and mesh-free interpolation capabilities: multivariate data interpolation between point clouds and grids; compactly supported radial basis functions; nearest-neighbor and moving least square implementations; support for standard finite-element shape functions and user-defined interpolants; common applications include conjugate heat transfer, fluid structure interaction, and mesh deformation.

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

Tasmanian (Toolkit for Adaptive Stochastic Modeling and Non-Intrusive Approximation)

Purpose: Constructs efficient surrogate models for high-dimensional problems and performs parameter calibration and optimization geared towards applications in uncertainty quantification (UQ).

Significance: UQ pertains to the statistical properties of the output from a complex model with respect to variability in multiple model inputs; large number of simulations are required to compute reliable statistics which is prohibitive when dealing with computationally expensive engineering models. A surrogate model is constructed from a moderate set of simulations using carefully chosen input values; analysis can then be performed on the efficient surrogate.

Sparse grids capabilities: surrogate modeling and design of experiments (adaptive multi-dimensional interpolation); reduced (lossy) representation of tabulated scientific data; high dimensional numerical quadrature; data mining and manifold learning.

DiffeRential Evolution Adaptive Metropolis (DREAM) capabilities: Bayesian inference; parameter estimation/calibration; model validation. global optimization and optimization under uncertainty.

URL: http://tasmanian.ornl.gov

ForTrilinos (Fortran Trilinos)

Purpose: ForTrilinos provides a seamless pathway for large and complex Fortran-based codes to access Trilinos without C/C++ interface code. This access includes Fortran versions of Kokkos abstractions for code execution and data management. To provide this functionality, this project developed a Fortran-targeted extension to the SWIG (Simplified Wrapper and Interface Generator) tool. Applied to Trilinos, it generates object-oriented Fortran 2003 interface code that closely mirrors the Trilinos C++ API.

Significance: The Exascale Computing Project (ECP) requires the successful transformation and porting of many Fortran application codes in preparation for ECP platforms. A significant number of these codes rely upon the scalable solution of linear and nonlinear equations. The Trilinos Project contains a large and growing collection of solver capabilities that can utilize next-generation platforms, in particular scalable multicore, manycore, accelerator and heterogeneous systems. Since Trilinos is written primarily in C++, its capabilities are not available to other programming languages. ForTrilinos bridges the gap between the needs of Fortran app developers and the capabilities of Trilinos. Furthermore, the technology used to generate the Fortran–C++ bindings in ForTrilinos is capable of exposing any number of C++ libraries to Fortran exascale app developers.

SWIG capabilities: ForTrilinos provides an inversion of control functionality that enables custom extensions of the Trilinos solvers implemented in downstream Fortran apps. Although this capability is not yet comprehensive, the goal of this project is to provide functional and extensible access Trilinos on next-generation computing systems. Several examples of ForTrilinos are being demonstrated within Fortran-based ECP codes to help them meet simulation goals and illustrate the technology to other Fortran-based ECP codes. Additionally, the SWIG technology underpinning ForTrilinos is being applied to other C++-based ECP ST subprojects to expose their capabilities to Fortran apps.

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

Key Challenges

ArborX: Search procedures to locate neighboring points, mesh cells, or other geometric objects require tree search methods difficult to optimize on modern accelerated architectures due to vector lane or thread divergence. A flexible interface for calling user kernels on a positive match as well as modifying traversal algorithms in a task-specific manner are crucial to achieving the best performance.

DTK: General data transfer between grids of unrelated applications requires many-to-many communication which is increasingly challenging as communication to computation ratios are decreasing on successive HPC systems. Maintaining high accuracy for the transfer requires careful attention to the mathematical properties of the interpolation methods and is highly application-specific.

Tasmanian: Extracting statistical information from a Tasmanian surrogate (or using the surrogate in a multi-physics simulation) requires the collection of a large number of samples, which is not feasible without GPU acceleration. The GPU accelerated surrogate evaluations require both custom kernels corresponding to the different types of basis functions as well as both sparse and dense linear algebra methods (BLAS level 2 and 3). Porting the capabilities and optimizing the performance across different divergent architectures is challenging.

ForTrilinos: Developing the interfaces to the C++ libraries that provide access to cutting-edge research, such as Trilinos, is of significant benefit to Fortran community. However, such interfaces must be well documented, sustainable and extensible, which would require significant amount of resources and investment. This is further complicated by the requirements to support heterogeneous platforms (e.g., GPUs) and inversion-of-control functionality. The manual approach to such interfaces has been shown to be unsustainable as it requires interface developers to have in-depth expertise in multiple languages and the peculiarities in their interaction on top of the time commitment to update the interfaces with changes in the library.

ForTrilinos addresses both the issue of reducing interface generation cost through investment in tool configuration and usage to make the process as automatic as possible, and the issue of providing the full-featured interface to Trilinos library, including access to manycore, accelerator and heterogeneous solver capabilities in Trilinos.

Solution Strategy

ArborX: ArborX builds on a MPI+Kokkos programming model to deploy to all DOE HPC architectures. Extensive performance engineering has yielded implementations that are both as performant in serial as state-of-the-art libraries while also expanding on the capability provided by other libraries by demonstrating thread scalability on both GPU and multi-core CPU architectures. Working with both synthetic as well as real data from applications (e.g., HACC) ensures wide performance testing coverage.

DTK: State-of-the-art, mathematically rigorous methods are used in DTK to preserve accuracy of interpolated solutions. Algorithms are implemented in a C++ code base with extensive unit testing on multiple platforms. Trilinos packages are used to support interpolation methods. Kokkos is used to achieve performance portability across accelerated platforms.

Tasmanian: The C++ kernels within Tasmanian (currently tuned for Nvidia Volta architecture) are templated exposing numerous performance tweaks and tuning parameters that can be adjusted to perform well on a corresponding AMD system. The kernels also need to be ported to DPC++/SYCL to allow for the utilization of Intel GPUs. Tasmanian requires a general GPU-BLAS interface that can utilize any of the accelerated backends, e.g., cuBlas, rocBlas, MKL and MAGMA.

ForTrilinos: ForTrilinos defines several SWIG-Fortran modules that generate Fortran-2003 interfaces to C++ Trilinos solver classes. ForTrilinos provides a "high-level" interface for applications to access nonlinear and eigenvalue solvers in addition to low-level Trilinos classes.

Recent Progress

ArborX: Collaboration with partner application ExaSky (WBS 2.2.3.02) resulted in significant advances for the in-situ density-based clustering algorithm (halo finding) using Nvidia GPUs.

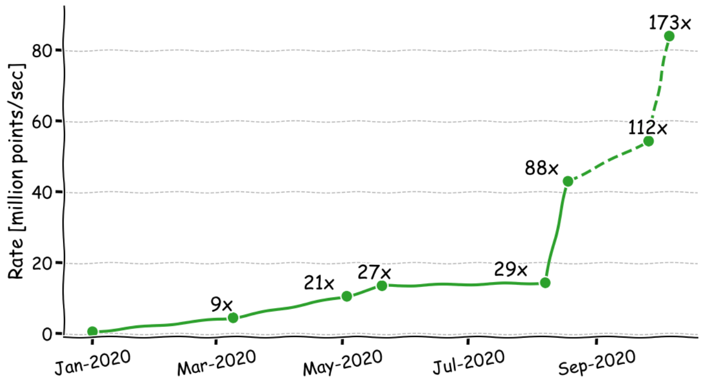


Figure 55: ArborX progress on halo finding algorithm on Nvidia Volta. The baseline is a serial implementation of CosmoTools. Numbers indicate speedup compared to the baseline. The solid lines show improvements that were already merged. Dashed lines show improvements that are in active development.

DTK: DTK's build system has been rewritten. DTK now depends on Trilinos instead of being built as an external package. DTK is now a separate package in spack. In the future this will allow a decoupling between Trilinos version and DTK version. A new spline interpolation method has been added.

Tasmanian: Work with partner application ExaStar (2.2.3.01) created a reduced representation of neutrino opacities used by the Thornado simulation software. The classical representation uses dense tables that do not fit in GPU memory and lead to unnecessary and expensive data movement for each time-step. The reduced representation by Tasmanian preserved the accuracy of the simulations and dramatically reduces the memory footprint by removing redundancies and exploiting smoothness in the data.

ForTrilinos: As with DTK, ForTrilinos now has an independent build system with Trilinos as a dependency. This improves robustness of the build and makes ForTrilinos available to app developers even if a system installation of Trilinos does not enable Fortran. ForTrilinos is now independently available through the Spack package manager. New ST libraries including Tasmanian have been wrapped with the SWIG-Fortran utility.

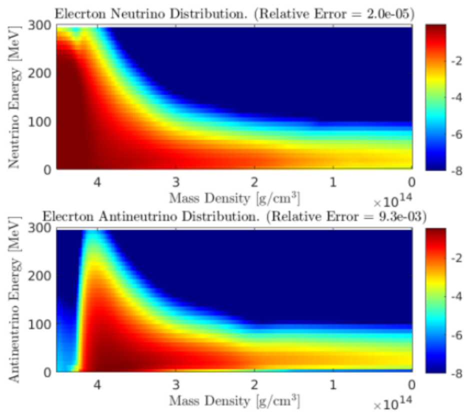


Figure 56: The resulting neutrino and antineutrino distributions in a deleptonization wave simulation using sparse grid opacities, which require only 6% of the memory used in the dense approach, with relative L^2 error less than 1%.

Next Steps

ArborX: Incorporate non axis-aligned bounding volumes to accommodate stretched inclined geometries such as those coming from wind turbine simulations from ExaWind (WBS 2.2.2.01). Further improve performance of density-based algorithms.

DTK: Continue performance engineering campaign and deploy in a variety of applications.

Tasmanian: Port the surrogate evaluation kernels to AMD and Intel GPUs and optimize the performance on the next generation architectures (including next generation Nvidia GPUs).

ForTrilinos: Extend ForTrilinos native Fortran interface documentation and prioritize Fortran app customer needs. Integrate SWIG into ECP ST projects that desire Fortran interfaces.

4.4 WBS 2.3.4 DATA & VISUALIZATION

End State: A production-quality storage infrastructure necessary to manage, share, and facilitate analysis of data in support of mission critical codes. Data analytics and visualization software that effectively supports scientific discovery and understanding of data produced by Exascale platforms.

4.4.1 Scope and Requirements

Changes in the hardware architecture of Exascale supercomputers will render current approaches to data management, analysis and visualization obsolete, resulting in disruptive changes to the scientific workflow and rendering traditional checkpoint/restart methods infeasible. A major concern is that Exascale system concurrency is expected to grow by five or six orders of magnitude, yet system memory and input/output (I/O) bandwidth/persistent capacity are only expected to grow by one and two orders of magnitude, respectively. The reduced memory footprint per FLOP further complicates these problems, as does the move to a hierarchical memory structure. Scientific workflow currently depends on exporting simulation data off the supercomputer to persistent storage for post-hoc analysis.

On Exascale systems, the power cost of data movement and the worsening I/O bottleneck will make it necessary for most simulation data to be analyzed in situ, or on the supercomputer while the simulation is running. Furthermore, to meet power consumption and data bandwidth constraints, it will be necessary to sharply reduce the volume of data moved on the machine and especially the data that are exported to persistent storage. The combination of sharp data reduction and new analysis approaches heighten the importance of capturing data provenance (i.e., the record of what has been done to data) to support validation of results and post-hoc data analysis and visualization. Data and Visualization is the title for Data Management (DM) & Data Analytics and Visualization (DAV) activities in the Exascale project.

Data management (DM) activities address the severe I/O bottleneck and challenges of data movement by providing and improving storage system software; workflow support including provenance capture; and methods of data collection, reduction, organization and discovery.

Data analytics and visualization (DAV) are capabilities that enable scientific knowledge discovery. Data analytics refers to the process of transforming data into an information-rich form via mathematical or computational algorithms to promote better understanding. Visualization refers to the process of transforming scientific simulation and experimental data into images to facilitate visual understanding. Data analytics and visualization have broad scope as an integral part of scientific simulations and experiments; they are also a distinct separate service for scientific discovery, presentation and documentation purposes, as well as other uses like code debugging, performance analysis, and optimization.

The scope of activities falls into the following categories:

- Scalable storage software infrastructure – system software responsible for reliable storage and retrieval of data supporting checkpointing, data generation, and data analysis I/O workloads
- Data collection, reduction, and transformation – enabling complex transformation and analysis of scientific data where it resides in the system and as part of data movement, in order to reduce the cost to solution
- Data organization and discovery – indexing and reorganizing data so that relevant items can be identified in a time- and power-efficient manner, and complex scientific data analysis can be performed efficiently on Exascale datasets
- In situ algorithms and infrastructure – performing DAV while data is still resident in memory as the simulation runs enabling automatic identification, selection and data reduction for Exascale applications.
- Interactive post-hoc approaches – on data extracts that produced in situ and support post-hoc understanding through exploration.
- Distributed memory multi-core and many-core approaches, for the portable, performant DM and DAV at Exascale.

4.4.2 Assumptions and Feasibility

- Scaling up traditional DM and DAV approaches is not a viable approach due to severe constraints on available memory and I/O capacity, as well as dramatically different processor and system architectures being at odds with contemporary DAV architectures.
- Simulations will produce data that is larger and more complex, reflecting advances in the underlying physics and mathematical models. Science workflows will remain complex, and increasing requirements for repeatability of experiments, availability of data, and the need to find relevant data in Exascale datasets will merit advances in workflow and provenance capture and storage.
- The expense of data movement (in time, energy, and dollars) will require data reduction methods, shipping functions to data, and placing functionality where data will ultimately reside.
- Solid-state storage will become cheaper, denser, more reliable, and more ubiquitous (but not cheap enough to replace disk technology in the Exascale timeframe). Exascale compute environments will have in-system nonvolatile storage and off-system nonvolatile storage in addition to disk storage. Applications will need help to make use of the complex memory/storage architectures.
- Disks will continue to gain density but not significant bandwidth; disks will become more of a capacity solution and even less a bandwidth one.
- Industry will provide parts of the overall data management, data analysis and visualization solution, but not all of it; non-commercial parts will be produced and maintained.
- This plan and associated costs were formulated based on the past decade of DOE visualization and data analysis activities, including the successful joint industry/laboratory-based development of open-source visualization libraries and packages (VTK, VisIt, and ParaView).

4.4.3 Objectives

Data management, analysis and visualization software must provide:

- production-grade Exascale storage infrastructure(s), from application interfaces to low-level storage organization, meeting requirements for performance, resilience, and management of complex Exascale storage hierarchies;
- targeted research to develop a production-grade in situ workflow execution system, to be integrated with vendor resource management systems, meeting science team requirements for user-defined and system-provided provenance capture and retention;
- production-grade system-wide data transfer and reduction algorithms and infrastructure, with user interface and infrastructure for moving/reducing data within the system, to be integrated with vendor system services and meeting science and national security team requirements; and
- production-grade metadata management enabling application and system metadata capture, indexing, identification, and retrieval of subsets of data based on complex search criteria and ensures that technologies target science and national security team requirements.
- targeted research to develop a production-grade in situ algorithms, to be integrated with open source visualization and analysis tools and infrastructure, meeting science team data reduction requirements
- targeted research to develop a production-grade algorithms for the new types of data that will be generated and analyzed on Exascale platforms as a result of increased resolution, evolving scientific models and goals, and increased model and data complexity.
- targeted research to develop a production-grade post-hoc approach that support interactive exploration and understanding of data extracts produced by in situ algorithms
- production-grade Exascale data analysis and visualization algorithms and infrastructure, meeting requirements for performance, portability and sustainability for evolving hardware architectures and software environments.

4.4.4 Plan

Productization of technologies is a necessary step for adoption, research-quality software is not enough. One approach we will take is to fund vendors of products in related areas to integrate specific technologies into their product line. When developing objectives for this activity, a focus was placed on the availability of products that deliver these technologies on platforms of interest. Activities can be separated into two categories:

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

Portions of the DAV software stack are being productized and supported by industry, which will help to control costs in the long term. Activities to achieve the DAV objectives are heavily dependent on developments across the Exascale project, and thus close coordination with other teams is essential. Close engagement with application scientists is crucial to the success of DAV, both in terms of understanding and addressing the requirements of science at scale and ensuring that computational scientists are able to adopt and benefit from the DAV deliverables.

Many objectives need initial research projects to define plausible solutions. These solutions will be evaluated and progressively winnowed to select the best approaches for the Exascale machine and the needs of science. Selected projects will continue to receive support to extend their research and development efforts to integrate their solutions into the open-source Exascale software stack.

4.4.5 Risks and Mitigations Strategies

There are specific risks identified for the Data and Visualization portfolio. These risks are tracked in the risk register .

- Application teams may continue to employ ad hoc methods for performing data management in their work, resulting in increased I/O bottlenecks and power costs for data movement. Application team engagement, working within the overall software stack, and input into Hardware Integration will be necessary if results are to be deployed, adopted, and significantly improve productivity.
- Despite funding vendor activities, industry partners may determine the market is insufficient to warrant meeting Exascale requirements.
- If vendor integration and targeted R&D activities are not closely coordinated, gaps will not be effectively identified and targeted, or successful R&D will not be integrated into industry products in the necessary timeframe.
- Vendors supplying data management solutions are likely to be distinct from Exascale system vendors. Additional coordination will be necessary, beyond DM productization, in order to ensure interoperability of DM solutions with specific Exascale platforms.
- Data management from an application perspective is tracked in one of the identified risks. Additionally, the software stack tracks several risks indirectly related to data management as well.
- Failure of scientists to adopt the new DAV software is a major risk that is exacerbated if the DAV software is research quality. Mitigating this risk depends on close engagement with domain scientists and supporting layers of the software stack through co-design activities, as well as investment in development and productization of DAV codes.
- Redundant efforts in domain science communities and within ASCR-supported activities such as SciDAC result in wasted resources. Communication and close coordination provide the best strategy for mitigation.

- Fierce industry and government competition for DAV experts creates a drain on laboratory personnel in DAV and makes lab hiring in this area difficult. Stable funding and a workforce development program would help to mitigate these risks.
- A skilled workforce is required for a successful Exascale project.

4.4.6 Future Trends

Graphics Architectures and Approaches Graphics architectures are improving in terms of raw computational power and through the addition of specialized libraries for accelerating ray-tracing, volume rendering, and denoising. Nvidia has added specialized hardware processing units for ray-tracing and machine learning to their GPU offerings. Intel has developed a suite of CPU accelerated libraries that support OpenGL (OpenSWR), ray-tracing (Embree, OSPRay), volume rendering (Open Volume Kernel Library) and de-noising (Open Image Denoise). From a visualization and rendering perspective, ray-tracing provides significantly improved rendered results over traditional scan-conversion based approaches. A near-term opportunity is to take advantages of such functionality for our rendering needs. Longer term, we will look into leveraging these hardware accelerated approaches to accelerate visualization and analysis tasks.

In Situ Analysis and Automation A key thrust of the Data and Visualization area is the focus on in situ analysis in order to filter important information as it is being generated by the simulations. In addition to our algorithmic and infrastructure efforts, automatic techniques and workflows must be developed to guide the overall in situ analysis process.

Workflows Slowly, more complex workflows are becoming a more significant component of the job mix on ECP-relevant platforms, partially driven by the increased use of these systems for machine learning applications. Workflows can drive degenerate use cases in the storage stack, such as the use of the file system for communication between tasks, when tools from outside the HPC community are adopted without change. Alternative approaches to enable communication between tasks exist but must be adapted to facility contexts, and technical roadblocks (e.g., difficulty in communicating between separate jobs) must be overcome.

AI AI applications will appear more frequently in the job mix. This impacts the requirements for data storage, as new classes of data become more prominent in application input datasets. It also impacts technologies for understanding application behavior, as these jobs are often not using MPI, a common assumption in tool sets. Finally AI-focused applications do not exhibit the common pattern of alternating phases of I/O and computation seen in simulation codes, driving a need for attention on methods of I/O optimization that do not rely on explicit collective I/O phases.

Networks Network architectures are still in flux, and specific new technologies such as Slingshot from Cray will bring new capabilities such as more advanced congestion detection and mitigation that change how networks will behave in the face of mixed communication and I/O traffic or the impact of communication-heavy applications on other applications in the system, etc. Assumptions regarding how I/O traffic fits into this picture may need to be reexamined. The libfabric interface for accessing networks appears to be the most promising portable interface for use outside of MPI, and teams will need to assess how to best use libfabric across platforms of interest as well as possibly advocating for specific capabilities in libfabric that fall outside of traditional MPI use cases, such as the common pattern of clients connecting and detaching from long-running services.

Object stores Facilities are planning deployments of non-POSIX storage solutions. One of the first of these will be the DAOS deployment on the A21 system at Argonne. The DAOS interfaces are available for teams to begin to understand, an HDF5 front-end for DAOS is available, and there are some examples of DAOS use for scientific codes. It is likely that the highest performance will come from applications directly using the DAOS APIs, and work to allow understanding of how these APIs are used would be beneficial.

Compression Compression will continue to play an important role in computation as a vehicle for addressing the explosion in size of datasets and outputs. Improved integration of compression capabilities in libraries supporting parallel I/O will continue to be a topic for further development, and techniques for allowing concurrent updates while compression is enabled specifically need more exploration. The use of lower precision data types has the potential of speeding up the visualization and analysis process as well as reducing data sizes without significantly degrading the accuracy of results.

Storage technologies and architectures Even in systems that will continue to employ POSIX file systems as the main ”scratch” store, the hardware on which these file systems are stored will be changing. For example, the Perlmutter system will provide a 30 PB nonvolatile storage tier using Lustre. The file system teams (e.g., Lustre team) will be working to maximize performance on these new storage back-ends, but simultaneously higher software layers must consider how this significant change impacts their assumptions about the relative costs of communication and data storage for common patterns of access.

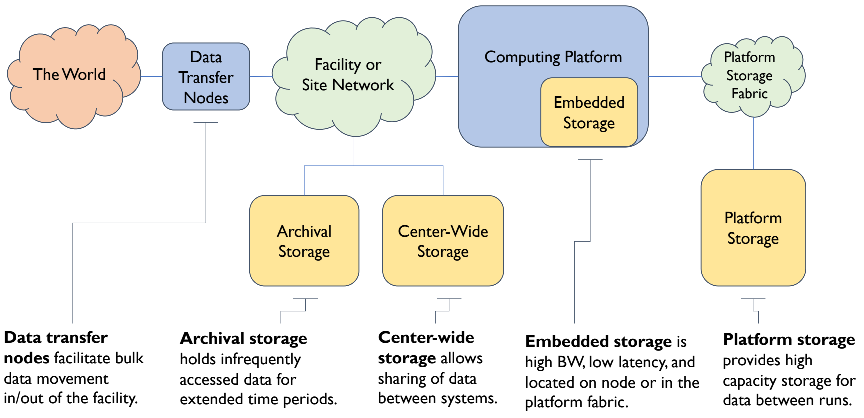

The World
Data Transfer Nodes
Facility or Site Network
Computing Platform
Embedded Storage
Platform Storage Fabric
Archival Storage
Center-Wide Storage
Platform Storage
Data transfer nodes facilitate bulk data movement in/out of the facility.
Archival storage holds infrequently accessed data for extended time periods.
Center-wide storage allows sharing of data between systems.
Embedded storage is high BW, low latency, and located on node or in the platform fabric.
Platform storage provides high capacity storage for data between runs.

Figure 57: A notional diagram of DOE facility storage resources. Not all systems have each role filled, and often additional network connections exist to accelerate specific data flows.

Figure 57 depicts a notional diagram of the storage resources surrounding a leadership class platform at a DOE facility. In this diagram, "platform" is the HPC system itself: Theta, Summit, Cori, Frontier, Perlmutter, Aurora, etc.

Note that this is a notional diagram. There are often additional connections to speed specific transfers, and some sites augment one resource while omitting another. Data transfer nodes provide access to data stored on facility storage from the outside world: Typically this is enabled using GridFTP, Globus Online, htar, or similar.

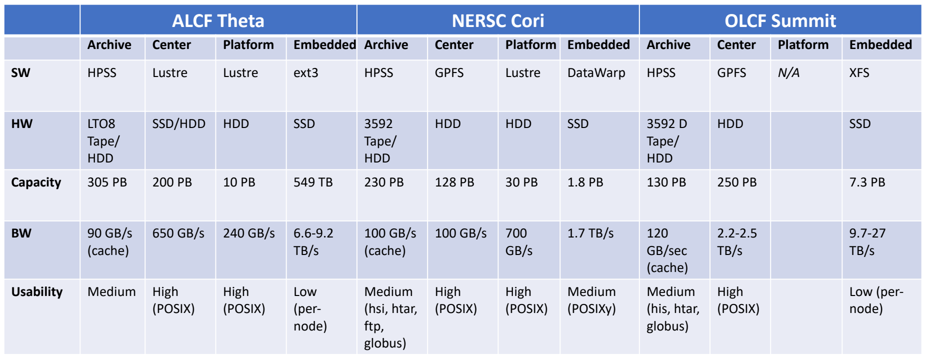
There are lots of roles that storage might play:

- *Archival storage* holds "cold" data. Currently tape is still the dominant media for archival storage, but disk is also used, both as cache and as permanent storage (sometimes spun down).
- *Center-wide storage* allows for easy data access between systems. This might include home directories and some other shared data volumes. Often performance is limited compared to platform storage.
- *Platform storage* is storage that is connected to a limited number of platforms in the facility and is meant to be a high-performance, high-capacity store for data that will be used for multiple runs. While disk drives are still used in some platform storage deployments, increasingly solid-state storage (e.g., SSDs) is being employed in this role.
- *Embedded storage* is located very close to the platform itself, either as node-local storage or tightly integrated into the fabric. Technologies used include NVMe and SSDs. Embedded storage that is available across the system is perhaps best thought of as a special kind of platform storage.

Table 12 captures the salient characteristics of the storage deployments for the current generation of systems: Theta, Cori, and Summit. These systems largely reflect trends in DOE storage over the last decade: HPSS archival storage coupled with a POSIX center-wide file system provided by GPFS or Lustre and backed by hard disk drives (HDDs). In two cases, a faster, platform-specific POSIX file system is deployed, while at OLCF the team chose to instead concentrate on a very high performance center-wide file system that is available on other resources as well.

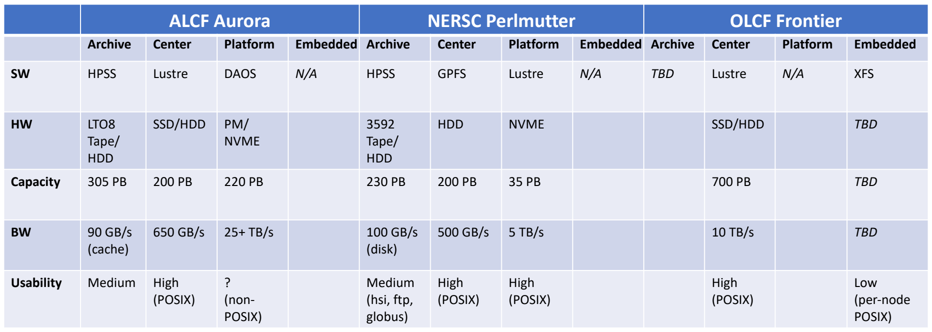
Of note in these systems are some embedded storage options that have provided the community with some early experiences with SSD storage. On Theta and Summit, local SSDs are available that have seen limited use by specific teams. On Cori, the DataWarp service allows for SSD-backed storage pools to be

Table 12: Storage system specifications for current platforms.



	ALCF Theta				NERSC Cori				OLCF Summit			
	Archive	**Center**	**Platform**	**Embedded**	**Archive**	**Center**	**Platform**	**Embedded**	**Archive**	**Center**	**Platform**	**Embedded**
SW	HPSS	Lustre	Lustre	ext3	HPSS	GPFS	Lustre	DataWarp	HPSS	GPFS	*N/A*	XFS
HW	LTO8 Tape/ HDD	SSD/HDD	HDD	SSD	3592 Tape/ HDD	HDD	HDD	SSD	3592 D Tape/ HDD	HDD		SSD
Capacity	305 PB	200 PB	10 PB	549 TB	230 PB	128 PB	30 PB	1.8 PB	130 PB	250 PB		7.3 PB
BW	90 GB/s (cache)	650 GB/s	240 GB/s	6.6-9.2 TB/s	100 GB/s (cache)	100 GB/s	700 GB/s	1.7 TB/s	120 GB/sec (cache)	2.2-2.5 TB/s		9.7-27 TB/s
Usability	Medium	High (POSIX)	High (POSIX)	Low (per-node)	Medium (hsi, htar, ftp, globus)	High (POSIX)	High (POSIX)	Medium (POSIXy)	Medium (his, htar, globus)	High (POSIX)		Low (per-node)

Table 13: Projected storage specifications for upcoming platforms.



	ALCF Aurora				NERSC Perlmutter				OLCF Frontier			
	Archive	**Center**	**Platform**	**Embedded**	**Archive**	**Center**	**Platform**	**Embedded**	**Archive**	**Center**	**Platform**	**Embedded**
SW	HPSS	Lustre	DAOS	*N/A*	HPSS	GPFS	Lustre	*N/A*	*TBD*	Lustre	*N/A*	XFS
HW	LTO8 Tape/ HDD	SSD/HDD	PM/ NVME		3592 Tape/ HDD	HDD	NVME			SSD/HDD		*TBD*
Capacity	305 PB	200 PB	220 PB		230 PB	200 PB	35 PB			700 PB		*TBD*
BW	90 GB/s (cache)	650 GB/s	25+ TB/s		100 GB/s (disk)	500 GB/s	5 TB/s			10 TB/s		*TBD*
Usability	Medium	High (POSIX)	? (non-POSIX)		Medium (hsi, ftp, globus)	High (POSIX)	High (POSIX)			High (POSIX)		Low (per-node POSIX)

allocated that are visible to an entire job or workflow. This functionality is close to the model that users are accustomed to, and the resource has seen significant use.

Table 13 captures the current, public understanding of storage characteristics for the next generation of systems: Aurora, Perlmutter, and Frontier. A number of observations can be made from this data. First, POSIX will remain the dominant (at least low-level) API for interacting with the primary storage resources (embedded, platform, and center-wide). Obtaining peak performance will likely remain a challenge for users, but with node counts somewhat stalled, storage software scalability is not further pressured. Second, DAOS, will appear as a POSIX alternative. DAOS is an Intel product that will provide a form of globally accessible key-value style of storage. However, a POSIX "veneer", or compatibility layer, will be provided that will give users an easy way to make use of the resource. With all the major ST I/O libraries already having the ability to implement alternative back-ends, the challenge will be more about how to persist things *outside* DAOS efficiently (e.g., converting back to POSIX files). Finally, there will still be a non-global, embedded storage resource on Frontier. The best ways to utilize this resource need to be studied, but systems like UnifyFS could play a role.

4.4.7 WBS 2.3.4.01 Data & Visualization Software Development Kits

DataViz SDK Overview

The Data & Visualization (DataViz) SDK aims to create a production-quality infrastructure necessary to manage, share, and facilitate data analysis of mission-critical codes at scale. The project focuses on community development and a commitment to success via quality improvement policies, better build and deployment processes, and the ability to use diverse, independently developed DataViz SDK projects, in combination, for data analysis and visualization problems.

The DataViz SDK's responsibility is to coordinate the disparate documentation, development, testing, deployment activities, and develop the necessary tooling and shared infrastructure to serve these goals. This coordination's resulting product is a unified set of usable, standardized, and interoperable packages ready for the upcoming exascale machines. We have designed the efforts to support the DataViz SDK to fit within the overarching goal to leverage and integrate data management, analysis, and visualization techniques developed across the ECP Software Technology ecosystem to support scientific discovery and understanding.

In addition, DataViz SDK provides a capability supporting collaborative analysis and visualization software development, helping independent teams accelerate the adoption of best practices, enabling interoperability of independently developed software, and improving developer productivity and sustainability of the software products.

Key Challenges

Scientists and engineers from various research cultures and significantly different software engineering maturity levels develop Data & Visualization software packages. In addition to the challenges outlined in Section 4.5.7, ECP Data & Visualization software packages will use different combinations of dependency software in various configurations. Visualization applications and libraries, in particular, utilize lower-level graphics libraries from an OpenGL stack needing reliably mapped to a diverse set of underlying hardware. These applications demand the deployment infrastructure support the appropriate combination of software and hardware-based rendering on NVIDIA, AMD, or Intel GPUs and accelerated offscreen rendering APIs like EGL. Requirements such as these place the DataViz SDK between the hardware teams and the analysis and visualization software developers preparing to run on new architectures yet delivered by ECP.

DataViz SDK software packages also have, on average, a much deeper dependency chain than is typical within HPC. As of this writing, an optimized set of twelve ECP DataViz SDK packages requires over 160 dependencies. Many packages share these dependencies, each with their own set of constraints. This combination presents a unique challenge to ensure compatibility, interoperability, and reliability of the entire stack as a whole, beyond the individual packages.

Solution Strategy

First, the DataViz SDK solution strategy involves pursuing usability, standardization, interoperability, and sustainability goals through a set of community policies to improve software practices. The DataViz SDK community policy tasks have required us to define a common terminology for effective communication.

Second, we leverage shared infrastructure, such as the Spack [1] package manager and CI testing at ECP facilities. We built the SDK release and delivery deployment goals on Spack as a unifying package manager, while our reliability and sustainability goals benefit from and leverage the facilities' CI testing infrastructure.

Finally, we define a set of spack meta-packages, ecp-vis-sdk and ecp-io-sdk, to enable the delivery of ECP targeted configurations of data and visualization packages through E4S. These meta-packages establish dependencies for the packages within the DataViz SDK, and serve as the backbone of our interoperability testing and deployment efforts.

- ecp-vis-sdk includes data analysis and visualization packages such as ASCENT, Catalyst, Cinema, ParaView, VTK-m along with data reduction and compression libraries like SZ, and ZFP.
- ecp-io-sdk includes input and output (I/O) services like ADIOS, Darshan, hdf5, parallel-netcdf, unifyfs, and veloc.

The packages contained in the two meta-packages represent software at different maturity and readiness for release. The early release strategy was to push product readiness for inclusion in the E4S releases by assisting packages with Spack packaging and CI testing. We will continue to evolve the maturity level and interoperability of the packages while preparing for subsequent releases.

Recent Progress

In pursuit of establishing a baseline set of software quality across the entire ECP ST area, the SDK projects have been collectively developing a set of community policies a given package must adhere to be an E4S member package. These community policies cover areas including publicly accessible documentation, mandatory spack packaging, testing practices, and other policy areas. The DataViz SDK has played an active role in this panel by proposing new policies, refining language on official policies, and soliciting community feedback. We are nearing a final set of initial guidelines required by the E4S project for packages to be accepted as a first-class E4S Member Package. These community policies intended to be an ongoing collaborative effort to elevate the standard of software quality, reliability, and sustainability for the entire ECP software ecosystem.

In addition, recent progress included the completion of the following three milestones during the fiscal year 2020.

STDV01-06 — Improved CI capabilities. During this milestone, we worked to ensure that all of the packages within the DataViz SDK were utilizing Continuous Integration in some way as part of their development workflow and software process. While the packages span a wide range of maturity and robustness of the software process, they all have incorporated at least a baseline CI capability (some far more extensive than others). Most leverage a public cloud CI like Travis or GitHub Actions, while others rely on internal resources from an internal GitLab or BitBucket ser er. We also improved the spack packaging for several of the SDK projects and integrated the remaining projects into the SDK meta-packages.

STDV01-08 — HDF Virtual Object Layer Architecture (VOL) documentation and ECP HPC-CI integration. During this time, we worked directly within the ADIOS and VTK-m projects to enable the newly developed ECP HPC-CPI capability. These two projects served as early adopters of the Gitlab-CI environment running directly on HPC resources at ORNL. The SDK coordinated directly with ORNL facilities personnel to debug the CI environment worked to identify issues other Data & Viz projects are likely to encounter, and develop the initial capability for ADIOS and VTK-m. From this work, the SDK can assist other data and visualization projects in implementing the ECP HPC CI capability through both guidance and best practice recommendation and direct technical development assistance.

STDV01-14 — Optimized Spack configurations and CI. The default configuration of most spack packages is intended to produce the most compatible version, but not necessarily the most optimal for large scale HPC. In particular, packages may have key features disabled by default essential to HPC, such as MPI support and FORTRAN language bindings. The two meta-packages were enhanced to ensure that an optimized configuration explicitly targets ECP target platforms for every direct data and visualization package dependency.

Next Steps

We highlight our next steps in the follow on project milestones.

STDV01-17 — Cross SDK CI testing. The focus of this release is to demonstrate successful interoperable CI testing. The DataViz SDK is building out a CI infrastructure to allow all ST products within the Data & Visualization focus area to be regularly built and tested with each other to ensure interoperability. This milestone is to have the CI system running with as many ST products in the DataViz SDK successfully building together, satisfying each other's dependencies.

STDV01-29 — Hardening to ensure SENSEI is deployable and reliable at scale. Establishing Spack recipes for the SENSEI software coupled with the hardening of the in transit code-base will go a long way in mitigating the risks associated with availability and defects for ECP applications.

HDF5

Overview

HDF5 is a data model, I/O library, and file format to store and manage data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O for high volume and complex data. HDF5 is portable and is extensible, allowing applications to evolve in their use of HDF5. The HDF5 software suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format. Numerous ECP applications use HDF5 or high-level libraries built on top of HDF5 (for example, netCDF-4, H5Part) for managing application's data. Therefore, HDF5 is a critical part of the Data & Visualization SDK. It is developed and maintained by a nonprofit organization, The HDF Group.

Under the Data & Visualization SDK effort, The HDF Group releases HDF5 new features that enhance workflow, productivity and ECP applications performance. Such features, for example, include ExaHDF5 productized Virtual Object Layer Architecture (VOL) and HDF5 VOL connectors, which allow ECP applications to access data on different storage devices, including Tiered Memory and to access data in various file formats. In this activity, we focus on:

- Identifying and prioritizing components of SDK that could benefit from current underutilized HDF5 features and newly released features and recommending changes to the identified components.
- Assisting other Data & Visualization SDK team, ADIOS, ExaIO and DataLib teams, in requirements gathering for integration with HDF5 APIs.
- Assuring that the latest released HDF5 software and non-integrated features developed under ExaIO (e.g., Async I/O VOL), are part of CI testing on the ECP platforms, including integrating HDF5 into the SDK CI testing framework (GitLab).
- Addressing any HDF5 related CI testing issues, in addition to any HDF5 bug or deficiently affecting parallel I/O performance, sustainability, and/or utility on ECP platforms.

The HDF Group is performing outreach activities as described here:

- Reaching out to the ECP science application teams that use or intend to use HDF5 and assess applications' usage of HDF5 or I/O needs, recommend best practices and existing HDF5 features to achieve scalable performance and to avoid I/O bottlenecks when using SDK components.
- Identifying necessary improvements to data organization in HDF5, to the usage of HDF5 library features, and assisting applications to implement identified improvements.
- Integrate ECP supported ZFP and SZ compression library with the HDF5 maintenance releases.
- Series of HDF5 Tutorials for new and advanced HDF5 users, and seminars for the HDF5 applications developers on the HDF5 best practices and performance.

Key Challenges

HDF5 is a complex software used not only to perform I/O, but also to manage complex data in one HDF5 file. Very often, developers of HDF5 applications are challenged to find the right balance between optimum I/O performance and data organization in the HDF5 file for further processing and sharing. It is unreasonable to expect scientists to go over more than 500 HDF5 C functions to find the right tuning knobs. As a result, some application teams continue using home-grown `ASCII` and binary formats and not taking advantage of HDF5 features and especially the features developed for ECP.

Solution Strategy

To lower the barrier for adopting HDF5 by ECP applications, The HDF Group has to provide quality HDF5 software that works on ECP platforms, integrate newly developed features into mainstream HDF5 and make them available to ECP applications promptly, educate HDF5 users on major HDF5 features, tuning tools and tuning techniques, and work closely with ECP applications teams on application tuning.

Recent Progress

During 2020 HDF5 *develop* branch and HDF5 1.12.0 and 1.10.7 maintenance releases were fully integrated with Spack and ECP CI testing using Gitlab on Ascent, Cori and several machines at LLNL. Test results are sent to The HDF Group public CDash.

We studied I/O access patterns of several ECP applications, including HACC, and summarized our findings in the white paper *"An I/O Study of ECP Applications"*. We summarize below the observations and some unexpected behaviors we found for each application along, with the suggestions on how to fix them. Detailed results and analysis can be found in the paper.

- **FLASH**: Unnecessary HDF5 metadata operations `H5Acreate()`, `H5Aopen()` and `H5Aclose()` are used during every checkpointing step. Those operations can be expensive, especially when running a large number of iterations. This can be easily fixed at the price of losing some code modularity.
- **NWChem**: File-per-process patterns are found for writing local temporary files. This is undesired and will cause a lot of pressures on parallel file systems for large scale runs. Conflicting patterns are found for the runtime database file, which can lead to consistency issues when running on non-POSIX file systems.
- **Chombo**: The Same file-per-process pattern is observed for Chombo too. Moreover, by default, Chombo uses independent I/O to write the final result to a shared HDF5 file. Depending on the problem scale and underlying file system configurations, collective I/O can further optimize the I/O performance.
- **QMCPack**: One unexpected pattern is found for checkpoint files. QMCPACK overwrites the same checkpoint file for each computation section. This could lead to an unrecoverable state if a failure occurred during the checkpointing step.
- **HACC-IO**: HDF5 can use different data layout to achieve similar MPI-IO access patterns. Stripe settings of the parallel file system have a big impact on the write performance. Also, the default metadata header can greatly slow down the write performance. However, carefully setting the alignment or metadata data block size, HDF5 can deliver similar performance as the pure MPI-IO implementation.

We hold several Webinars and Tutorials for HDF5 users. The recordings and corresponding materials are available from The HDF Group Website. Our goal was to introduce new and experienced HDF5 application developers to HDF5 tuning techniques and tools such as Darshan and Recorder to identify I/O performance bottlenecks. One of the Webinars was devoted to showing how HACC can achieve highly scalable performance when using HDF5. Setting right HDF5 metadata block size and Lustre or GPFS file system parameters allowed to match native HACC's MPI I/O approach when writing to a shared HDF5 file. Along with giving the Webinar: An Introduction to HDF5 in HPC Environments we created hands-on materials for the HDF5 Parallel Tutorial that is available from the GitHub repository. The Tutorial provides a quick start with parallel HDF5 and shows major techniques to get a good performance. We also created a hands-on Tutorial on how to troubleshoot HDF5 performance. Both Tutorials were presented at the second HDF User Group Meeting on October 13-16, 2020.

Next Steps

The HDF Group activities will continue in two areas: productization of the HDF5 features developed for ECP applications and outreach.

The HDF Group will continue working with the ExaIO, ADIOS and DataLib teams enhancing the HDF5 library and bringing HDF5 VOL connectors developed for ECP applications to production quality. We will also continue integrating developed features into HDF5 maintenance releases and CI testing. We are working to add CI testing to Theta and Summit.

ZFP and SZ HDF5 compression filters will be made more visible through HDF5 filter packages added to Spack and integrated with CI testing on the ECP systems. We plan to release HDF5 1.12.1 and integrate it with Spack and ECP CI testing. The release will address HDF5 library's issues discovered by the ExaIO team during the development of the Asynchronous HDF5 VOL connector and the DataLib team when developing the HDF5 logging VOL connector. We will also integrate a subfiling feature developed by the ExaIO team into the HDF5 maintenance releases.

We will continue working with the ECP HDF5 applications teams on I/O performance, and we will give Tutorials and Webinars, and create additional documentation on efficient usage of HDF5 in the ECP HPC environment.

4.4.8 WBS 2.3.4.09 ADIOS

Overview The Adaptable I/O Systems, ADIOS [160, 161], is designed to tackle I/O and data management challenges posed by large-scale computational science applications running on DOE computational resources. ADIOS has dramatically improved the I/O performance of Petascale applications from a wide range of science disciplines, thereby enabling them to accomplish their missions. The ADIOS ECP project is working on goal of transforming the ADIOS 1.x version, which has been used successfully on Petascale resources into a tool that will efficiently utilize the underlying Exascale hardware, and create a community I/O framework that can allow different ECP software to be easily "plugged" into the framework. The cornerstone of this project are to 1) efficiently address Exascale technology changes in compute, memory, and interconnect for Exascale applications; 2) develop a production-quality data staging method to support Exascale applications and software technologies that require flexible in situ data reduction and management capabilities; and 3) use state of the art software engineering methodologies to make it easier for the DOE community to use, maintain, and extend ADIOS. More precisely, our aim is to develop and deploy a sustainable and extensible software ecosystem. To make this into an ecosystem (rather than a point solution),this effort must result in an infrastructure than can be used effectively, customized, and shared among a variety of users, Exascale applications, and hardware technologies. Technically, we are achieving this goal by: refactoring ADIOS with the goal of improving modularity, flexibility, and extensibility by using C++; and extending, tuning, and hardening core services, such as I/O and staging that supports Exascale applications, architectures, and technologies.

Key Challenges The core challenge of ADIOS is in its name – adaptability. In order to present a uniform user interface while also being able to harness the performance improvements available in the wide variety of storage and interconnect capabilities, the internal structure of the ADIOS framework must address a number of portability, functionality, and performance tuning challenges. The internals should be constructed so that with no more than a small flag or runtime configuration a science code can move from doing I/O into a large Lustre parallel file system (with automatic calculation of file striping and number of files per directory) to utilizing burst buffer storage (with controls for delayed synchronization between the buffer and an archival store) or feeding the data directly into a concurrent application

The challenge of supporting hardware portability and runtime performance tuning also impose a third related challenge for software engineering of the system. In order for the code to be sustainable in the long term, while also offering guarantees of service to the end user, requires special attention to the architecture of the code base. The consequences of trying to address these three challenges, hardware portability, runtime performance, and sustainable engineering, have driven our approach and deliverable schedule for ADIOS in ECP.

Solution Strategy The ADIOS effort has two primary thrusts:

1. **Scalable I/O:** ADIOS has a data format designed for large scale parallel I/O and has data transport solutions to write/read data from/to the storage system(s) efficiently.

2. **Scalable data staging support:** ADIOS includes data transport solutions to work on data in transit, that is, to move data memory-to-memory, from one application stage to another without using file system I/O.

The challenges of portability and performance apply for both of these thrusts; to a certain extent, the third challenge around software engineering emerges from the need to support these two very different categories under a single user interface. Capitalizing on the experiences and performance expertise from our initial ADIOS platform, the ECP project wraps and extends this functionality to make it more sustainable, maintainable, and hopefully also more approachable for a wide community of users and developers. The project approach focuses on doing deep dives with end scientist users and developers in order to make sure that the computer science development process leads to specific, verifiable results that impact the customers.

Recent Progress A new version of the Application Programming Interface unifies staging I/O and file I/O [162], and the new, object-oriented, code framework [163] supports writing and reading files in two different file formats (ADIOS BP format and HDF5 format) and in situ with different staging implementations for various use cases. The new framework focuses on sustainable development and code reusability. The team also created the new scalable staging transport learning from the many lessons from using ADIOS for data staging and code coupling by applications in the past. As can be seen in Figure 58, this past experience with methods and deep science engagements has led to demonstrations at leadership computing scale (on Titan and Summit).

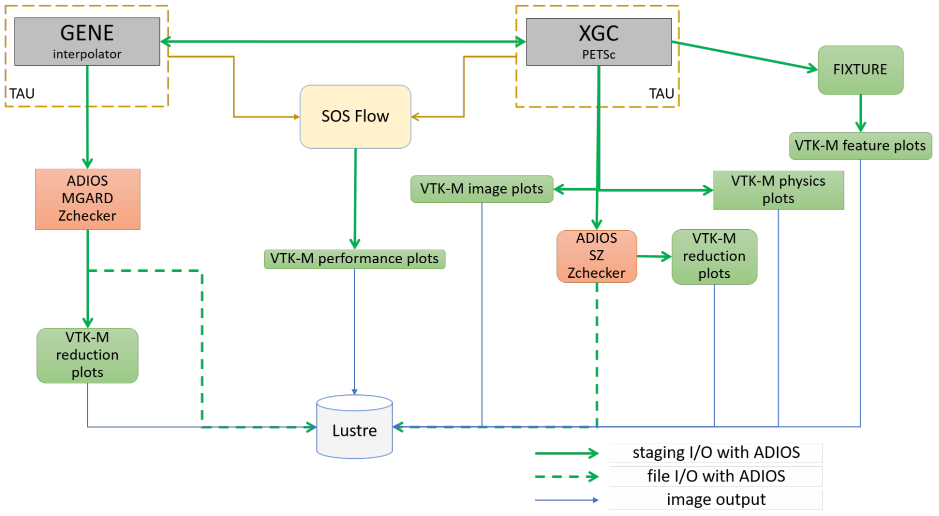


Figure 58: An example of using ADIOS to support ECP science. This sketch represents the demonstration at the February 2018 ECP Meeting, which featured WDM Fusion, CODAR, ADIOS, and other joint ECP activities. Note that all of the green arrows in the figure represent data communication or storage handled by the ADIOS infrastructure.

The new design focuses on stability and scalability so that applications can rely on it in daily production runs just as they have relied on the high performance file I/O of ADIOS. The new code base is governed with state-of-the art software development practices, including GitHub workflow of Pull-Requests with reviews, continuous integration that enforces well-tested changes to the code only, and nightly builds to catch errors on various combinations of architecture and software stack as soon as possible. Static and dynamic analysis are integrated to the GitHub workflow to catch errors before they cause trouble. Code coverage tools also help with increasing code quality. The team has access to and the code is continuously tested on DOE machines (Summit, Cori and Theta) using several ECP application codes and realistic science simulation setups (e.g. for WDMApp, E3SM-MMF and EXAALT application setups).

For interoperability with the other main I/O library used in the ECP program, HDF5, we have added compatibility in various ways. ADIOS has an engine to write/read HDF5 files using the original HDF5 library linked with ADIOS. A user can just change an option to switch from ADIOS-BP output to HDF5 output. On the other hand, ADIOS provides an HDF5-VOL layer, so that an HDF5 application can choose ADIOS as the underlying driver to write ADIOS-BP files from an application using HDF5.

Next Steps In the fifth year of the project we will be focusing on some special application cases, where the internal metadata of the ADIOS data representation leads to performance problems. Notably, the E3SM-MMF and WarpX applications need a better management of ADIOS metadata and blocks of the simulation data to achieve high write and read performance at scale. We will also have effort to prepare ADIOS for the exascale machines, Frontier and Aurora.

4.4.9 WBS 2.3.4.10 DataLib

Overview The Data Libraries and Services Enabling Exascale Science (DataLib) project has been pushing on three distinct and critical aspects of successful storage and I/O technologies for ECP applications: enhancing and enabling traditional I/O libraries used by DOE/ECP codes on leadership platforms, establishing a nascent paradigm of data services specialized for ECP codes, and working closely with facilities to ensure the successful deployment of our tools. In FY20-23 we plan to continue to focus on these three complementary aspects of storage and I/O technologies, adjusting in response to changing needs and bringing these three aspects together to provide the most capable products for end users. DataLib activities ensure that facilities have key production tools, including tools to debug I/O problems in ECP codes; enable multiple I/O middleware packages through Mochi and ROMIO; and will provide high performance implementations of major I/O APIs in use by ECP codes.

We strongly support ECP management's shift of focus towards **Hierarchical Data Format (HDF)**. In response to ECP guidance to prioritize the HDF5 API, we propose to emphasize enhanced HDF5 capabilities for ECP codes on current and future DOE leadership platforms, strengthening HDF as a core technology for the future. We propose to shift our focus away from ROMIO and PnetCDF development work to enable rapid progress on this topic. We will continue to support the use of Mochi tools for development of data services and I/O middleware, including assisting other ECP AD, ECP ST, and vendor teams in providing the best storage services possible for ECP applications. We will also continue to work closely with the facilities to ensure the availability and quality of our tools on critical platforms.

The **Darshan** I/O characterization toolset is an instrumentation tool deployed at facilities to capture information on I/O behavior of applications running at scale on production systems. It has become popular at many DOE facilities and is usually "on by default". Darshan data dramatically accelerates root cause analysis of performance problems for applications and can also (in some cases) assist in correctness debugging. Our work in this project focuses on extending Darshan to new interfaces and ensuring readiness on upcoming platforms.

The **ROMIO** and **Parallel netCDF** (PnetCDF) activities focus on existing standards-based interfaces in broad use, assisting in performance debugging on new platforms and augmenting existing implementations to support new storage models (e.g., "burst buffers"). In addition to being used directly by applications, ROMIO and PnetCDF are also indirectly used in HDF5 and netCDF-4. Our work is ensuring that these libraries are ready for upcoming platforms and effective for their users (and ready as alternatives if other libraries fall short).

The **Mochi** and **Mercury** software tools are building blocks for user-level distributed HPC services. They address issues of performance, programmability, and portability in this key facet of data service development. Mercury is being used by Intel in the development of their DAOS storage service and in other data service activities, while within ECP the HXHIM and UnifyCR projects also have plans to leverage these tools. In addition to working with these stakeholders and ensuring performance and correctness on upcoming platforms, we are also working with ECP application teams to customize data services for their needs (e.g., memoization, ML model management during learning). These are supporting tools that are not represented as official products in the ECP ST portfolio.

Key Challenges Each of these subprojects has its own set of challenges. Libraries such as HDF, ROMIO, and PnetCDF have numerous users from over a decade of production use, yet significant changes are needed to address the scale, heterogeneity, and latency requirements of upcoming applications. New algorithms and methods of storing data are required. For Darshan, the challenge is to operate in a transparent manner in the face of continuing change in build environments, to grow in functionality to cover new interfaces while remaining "lean" from a resource utilization perspective, and to interoperate with other tools that use similar methods to hook into applications. Mochi and Mercury are new tools, so the challenge in the context of these tools is to find users, adapt and improve to better support those users, and gain a foothold in the science community.

Solution Strategy *HDF enhancement.* HDF is the most popular high-level API for interacting with storage in the DOE complex, but users express concerns with the current The HDF Group (THG) implementation.

We propose to perform an independent assessment and systematic software development activity targeting the highest possible performance for users of the HDF5 API on ECP platforms of interest.

Directly supporting ECP applications and facilities. We currently have ongoing interactions with E3SM (PnetCDF), CANDLE (FlameStore/Mochi), and ATDM/Ristra (Quantaii/Mochi), and we routinely work with the facilities as relates to Darshan deployments. Our work with these teams is targeted on specific use cases that are inhibiting their use of current pre-exascale systems, such as E3SM output at scale using the netCDF-4/PIO/PnetCDF preferred code path. We will continue to work with these teams to address concerns, to maintain portability and performance, and may develop new capabilities if needs arise.

Supporting data services. Mochi framework components are in use in multiple ECP related activities, including in the UnifyCR and Proactive Data Containers (PDC) in ExaHDF5 (WBS 2.4.x) and in the Distributed Asynchronous Object Storage and other services in the Intel storage software stack. The VeloC and DataSpaces teams (WBS 2.4.x and x.y.z as part of CODAR, respectively) are also strongly considering adoption of our tools. Mochi components enhance the performance, portability, and robustness of these packages, and our common reliance on Mochi components means that as Mochi improves, so do all these users.

Integration and Software QA. DataLib has actively pursued integration with the ECP ST software stack through the development and upstreaming of Spack packages and the development and deployment of automated testing for DataLib technologies, so we are already well positioned in this aspect of our work. We anticipate this effort to continue throughout the FY20-23 timeframe, with the addition of pull requests submitted to THG to upstream HDF5 enhancements and effort applied to address identified issues in our technologies as appropriate.

Recent Progress *STDM12-22:* Deliver Darshan/HDF and CAR input, establish calls with Mochi users. Discussion occurred with ExaHDF team regarding their understanding of interesting HDF access characteristics. Basic Darshan capabilities for tracking HDF dataset use have been implemented in a (public) branch of Darshan for inclusion in a future release.

Unify, DataSpaces, and VeloC teams were contacted regarding regular communication. All teams were interested in a quarterly "open call" for all Mochi users. The first of these is planned for April and will be advertised on the mochi-devel mailing list. DataSpaces and VeloC teams were also interested in regular one-on-one calls. We will be meeting with the VeloC team in person next week to discuss cadence of these. We have the first one-on-one call with the DataSpaces team on Feb. 14. B. Robey is in touch with Ristra and ExaFEL.

STDM12-23: Analyze HDF use and improve Mochi services for ECP applications. Characterize performance and overhead of HDF use in specific ECP codes: we have been focusing on the FleCSI synthetic I/O benchmark developed for Ristra. A number of improvements have been made as a result of this work, including adjusting to use of collective metadata operations under HDF5, which help performance of this application use case. This exercise has led us to further consider how to improve how we present the new HDF5 data that can be captured as a result of our prior work (STDM12-22).

A synthetic benchmark has been developed that stores FleCSI data for Ristra in the HDF format. This benchmark is being used as a test environment for new output options for Ristra. The code, known as the flecsi-hdf5proxy, is available in the flecsi-incubator project. Our work in this benchmark has formed the basis for the Ristra HDF I/O design. Departing from the traditional approach of developing a proxy application for an already implemented capability, the initial version of the flecsi-hdf5proxy was developed prior to any implementation within Ristra codes. This allowed rapid design space exploration prior to implementation in a production code base and a more rapid development cycle in Ristra.

Early in the year, we initiated discussions with the CODAR team building Chimbuko, an effort to create a provenance and performance analysis service for HPC. Through discussion on their use case, we determined that a new Mochi microservice was needed. The Sonata microservice has been developed to provide convenient storage and processing of their JSON record data.

STDM12-24: Improve HDF use in ECP code, deliver report on Mochi use and CAR input. Regarding HDF performance, a summary was compiled, including some discussion of PnetCDF performance, which is still critical to E3SM. Additional notes on improving HDF performance can be found in our meeting notes.

Regarding Mochi use, feedback was compiled from our many users into a Mochi Customer Responses document, available in Jira. Regarding Ristra, the merged HDF proxy is in github as part of the FleCSI incubator.

Regarding CAR input, we have gathered initial data from ALCF, NERSC, and OLCF and identified personnel through which updates can be obtained as we get closer to the next CAR deliverable.

STDM12-25: Deliver design of HDF5 VOL plug-in and improve I/O capabilities for ECP application. HDF5 VOL design is complete, and we have an initial prototype with which we are performing early performance testing. The design document and developer notes will be attached to the end of this report for convenience.

As a reminder, our HDF VOL implementation layers on top of the "native" VOL implementation provided by HDF. We capture write operations as a log of changes, and currently we persist both the description of the writes (in our documentation simply the "metadata", stored in the "metadata table") and the contents of the writes (in our documentation the "log data", stored in the "log dataset") as HDF datasets. Additionally, another table (the "offset table") stores the offsets of specific datasets in the metadata table, allowing one to skip over unrelated datasets when looking for specific log entries. We're investigating the storage of metadata in memory, for performance reasons.

I/O Improvement in E3SM. We have previously extracted an I/O kernel from the E3SM code, and we are now working on a branch of this code that can write directly into HDF51. As background, the netCDF4 API is missing a key capability to describe I/O to multiple datasets as a single operation, and this limits the performance of netCDF4 for many scientific codes regardless of the underlying I/O API being used (e.g., PnetCDF, HDF). The E3SM team is aware of this deficiency, but to our knowledge there is no plan to address it at this time. Meanwhile, the HDF Group has been looking at multi-dataset writes of this type, although it is unclear when the capability might be made part of a production release. Never the less, working around the netCDF4 deficiencies allows us to isolate HDF performance/API challenges from the higher-level netCDF4 ones, and allows us to better explore our HDF VOL implementation.

I/O Improvement in xRAGE. Many applications use HDF5 to write extremely large single files used for checkpoint restart and/or data analysis. Even on pre-exascale systems such as Sierra it can be difficult to achieve high performance (bandwidth) for these workloads. Applications writing single shared files must carefully align I/O operations to avoid file locking overheads and balance I/O operations across multiple writers to ensure high performance. To help diagnose issues and optimize I/O in production Sierra applications the DataLib team has been working closely with LANL scientists to employ Darshan analysis capabilities that capture more detailed information about how HDF5, MPI-IO and POSIX are being used by the application and lower level software stacks. LANL has begun using Darshan on Sierra for a variety of applications including the xRAGE application and have worked with the DataLib team to identify potential bottlenecks in this and other applications.

Next Steps Our plan for FY21 includes: *STDM12-32:* Evaluate HDF prototype and build new Darshan capabilities. We will build a DAOS module and continue evaluation of HDF VOL prototype, adjusting design in response.

STDM12-33: Hold Mochi boot camp and release HDF VOL prototype. We will hold a Mochi training session, likely at the Annual Meeting, and we will release the HDF VOL prototype with a Spack package for others to work with.

STDM12-34: Enhance HDF VOL implementation and evaluate Darshan and Mochi performance. We will revisit Darshan overheads on flagship platforms to ensure correctness and low-overhead operation, and we will incorporate new enhancements into the HDF VOL implementation for further evaluation. We will also perform a performance evaluation of key Mochi use cases on available test hardware.

STDM12-35: Deliver UCX plug-in for Mercury in support of ECP applications and services. We will provide a tested UCX plug-in for Mercury, tuned for performance on ECP relevant platforms (e.g., Summit test systems), with appropriate nightly tests.

4.4.10 WBS 2.3.4.13 ECP/VTK-m

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

One of the biggest recent changes in high-performance computing is the increasing use of accelerators. Accelerators contain processing cores that independently are inferior to a core in a typical CPU, but these cores are replicated and grouped such that their aggregate execution provides a very high computation rate at a much lower power.

Current and future CPU processors also require much more explicit parallelism. Each successive version of the hardware packs more cores into each processor, and technologies like hyper threading and vector operations require even more parallel processing to leverage each core's full potential.

VTK-m is a toolkit of scientific visualization algorithms for emerging processor architectures. VTK-m supports the fine-grained concurrency for data analysis and visualization algorithms required to drive extreme scale computing by providing abstract models for data and execution that can be applied to a variety of algorithms across many different processor architectures.

The ECP/VTK-m project is building up the VTK-m codebase with the necessary visualization algorithm implementations that run across the varied hardware platforms to be leveraged at the Exascale. We will be working with other ECP projects, such as ALPINE, to integrate the new VTK-m code into production software to enable visualization on our HPC systems.

Key Challenges The scientific visualization research community has been building scalable HPC algorithms for over 15 years, and today there are multiple production tools that provide excellent scalability. However, our current visualization tools are based on a message-passing programming model. More to the point, they rely on a coarse decomposition with ghost regions to isolate parallel execution [164, 165]. However, this decomposition works best when each processing element has on the order of a hundred thousand to a million data cells [166] and is known to break down as we approach the level of concurrency needed on modern accelerators [167, 168].

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

Solution Strategy The ECP/VTK-m project leverages VTK-m [169] to overcome these key challenges. VTK-m has a software framework that provides the following critical features.

1. **Visualization building blocks:** VTK-m contains the common data structures and operations required for scientific visualization. This base framework simplifies the development of visualization algorithms [170].

2. **Device portability:** VTK-m uses the notion of an abstract device adapter, which allows algorithms written once in VTK-m to run well on many computing architectures. The device adapter is constructed from a small but versatile set of data parallel primitives, which can be optimized for each platform [171]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [172, 173, 174]. Within the device adapter we are leveraging Kokkos [175] to rapidly port to ECP hardware.

3. **Flexible integration:** VTK-m is designed to integrate well with other software. This is achieved with flexible data models to capture the structure of applications' data [176] and array wrappers that can adapt to target memory layouts [177].

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

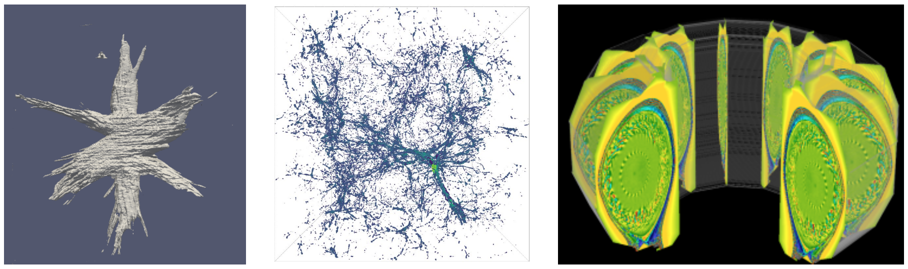


Figure 59: Examples of recent progress in VTK-m include (from left to right) optimized structured grid contouring, contouring of extended cell types, and representation of an extruded cell set.

Recent Progress The VTK-m project is organized into many implementation activities. The following features have been completed in the FY20 fiscal year.

- **VTK-m Releases:** VTK-m 1.5 was released in October 2019.
- **Kokkos:** Device adapters in VTK-m can now leverage the Kokkos programming model to more rapidly port to ECP hardware.
- **Improved Contouring:** VTK-m now implements the Flying Edges implementation of structured grid contouring [178]. The contouring in VTK-m, demonstrated in Figure 59, is now measured as one of the fastest implementations in existence. VTK-m has also contour support for expanded cell types to support more mesh types, also demonstrated in Figure 59.
- **Data Control Thread Safety:** The initial implementation of VTK-m assumed that all control would be on a single thread and all parallelism would be handled internally. However, several VTK-m customers need to launch GPU algorithms from multiple different threads. The internal management of array data has been redesigned to safely manage data from multiple control threads.
- **Spack Package:** Spack [179] is the package manager used to distribute the ECP ST software to the ECP platforms. The VTK-m package in Spack has been updated to the latest version of VTK-m.
- **Random Numbers:** Many algorithms rely on pseudo-random numbers. VTK-m now uses Salmon, et al's algorithm [180] to provide "random" arrays that allow algorithms to use numbers with random properties that are correct both within a thread and across threads of execution.
- **Extruded Cell Sets:** ECP's XGC simulation code uses an extrusion of a surface mesh for its data representation as demonstrated in Figure 59. A similar representation is added to VTK-m for zero-copy data ingestion.

Next Steps Our next efforts include:

- **Demonstrate VTK-m on Pre-Exascale Hardware:** With pre-exascale hardware available for O21 (Tulip) and A21 (Iris), VTK-m will be demonstrated on these platforms. For FY21 we expect to be able to compile VTK-m, run the regression tests, and use the benchmarking code to compare performance.
- **Support New Compiler Types:** VTK-m uses CMake to support cross-platform compilation. Using CMake with compilers that are not directly supported can be challenging. CMake will be updated for new compiler types as necessary, and VTK-m's build system will be similarly updated.
- **Resource Management:** Add general mechanisms to VTK-m that allow resource management (such as selecting which GPU to use when multiple GPUs are available on a node). Providing a general interface prevents code using VTK-m from having to use device-specific API's to control the resource.

4.4.11 WBS 2.3.4.14 VeloC: Very Low Overhead Checkpointing System

Overview. The VeloC-SZ project aims to provide VeloC, a high-performance, scalable checkpoint/restart framework that leverages multi-level checkpointing (the combination of several resilience strategies and heterogeneous storage) to ensure that ECP applications run to completion with minimal performance overhead. It delivers a production-ready solution that increases development productivity by reducing the complexity of having to deal with a heterogeneous storage stack and multiple vendor APIs. VeloC offers a client library that can be used by the applications to capture local application states, which are then coordinated and persisted using a resilience engine. VeloC runs the resilience engine asynchronously, which overlaps a large part of the checkpointing with the application runtime, thereby reducing its overhead. An overview of the architecture of VeloC is depicted in Figure 60.

VeloC has been released and shows significant lower checkpointing overhead for several ECP applications, such as HACC, LatticeQCD, EXAALT.

Key Challenges. VeloC faces several key challenges:

I/O bottlenecks: applications typically employ simple checkpoint-restart mechanisms to survive failures that directly use a parallel file system. With diminishing I/O bandwidth available per core, this leads to high checkpointing overhead and is not sustainable.

Deep heterogeneous storage: To compensate for diminishing parallel file system I/O bandwidth per core, the storage stack is becoming increasingly deeper and heterogeneous: node-local NVRAM, burst buffers, key-value stores, etc. However, the variety of vendors and performance characteristics make it difficult for application developers to take advantage of it.

Restart-in-place: a majority of failures affect only a small part of the nodes where the job is running. Therefore, reusing the surviving nodes to restart from the latest checkpoint immediately after a failure is more efficient than submitting a new job, which may wait in the batch queue.

Portability and robustness: applications need to run on a variety of supercomputing architectures, each featuring distinct capabilities. Their critical data structures that need to be checkpointed are constantly growing in size and complexity. Therefore, a flexible checkpointing solution is needed that can adapt to a variety of scenarios and configurations without sacrificing performance and scalability.

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

Multi-level checkpointing: is based on the idea that a majority of failures can be mitigated without involving the parallel file system: node-local checkpoints can be used to recover from software bugs, replication/erasure coding can be used to recover from most combinations of node failures. This reduces the frequency of checkpointing to the parallel file system and therefore the I/O bottlenecks.

Asynchronous mode: once a node-local checkpoint has been written, applications do not need to wait for replication, erasure coding or writes to the parallel file system: these can be applied in the background, while the application continues running. However, in this case, it is important to minimize interference with the application execution.

Transparent use of heterogeneous storage: we developed several techniques that can leverage a variety of local storage (in-memory file systems, flash storage) and external storage (burst buffers, key-value stores, parallel file systems) options. These techniques select the best available storage options, tune them with the optimal parameters and leverage any vendor-specific API if needed to transfer data.

Job scheduler integration: to implement restart-in-place, we have developed a series of scripts that interact with a variety of job schedulers to run jobs with spare nodes, continue execution on failures, restart on the surviving nodes and spares using the fastest possible recovery strategy (which ideally avoids reading checkpoints from the parallel file system). This is transparent to the users.

Declarative API and automated serialization: we offer a simple API that enables users to either manually capture checkpoints into files or to define memory regions that are automatically serialized into checkpoint files.

Modular design: applications link with a client library that is responsible to manage local checkpoints, while a separate engine is responsible to employ the rest of the resilience
strategies as plugable modules. This simplifies the implementation of the asynchronous mode, it enables users the flexibility to choose any combination of resilience strategies, as well as, to customize their checkpointing pipeline (e.g., add new post-processing operations such as analytics or compression).

Recent Progress We met and closely collaborated with several ECP application teams in an effort to address their checkpointing needs. Most of our current efforts involve the HACC, LatticeQCD, and EXAALT teams. Starting from our previous efforts to isolate the checkpointing code into an optional plugin within the application (which is notably the case of HACC, where we integrated VeloC as a CosmoTools plugin), we developed an automated deployment for the components of VeloC, which has two advantages: (1) it eliminates the need to change the application deployment scripts to launch the active backend; (2) it enables the application to decide the VeloC configuration dynamically at runtime.

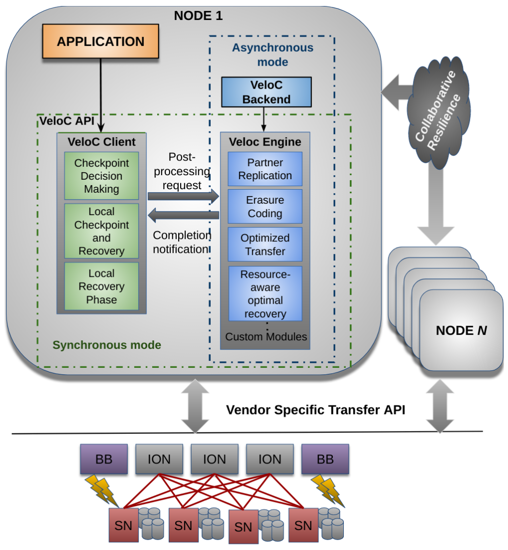


Figure 60: VeloC: Architecture

Furthermore, we added several new capabilities. First, we have added a *checksumming* module that verifies the integrity of the checkpoints This is done asynchronously to minimize the overheads of using this capability. Second, we have refactored the control plane that facilitates the communication between the VeloC clients and the active backends to include two new alternatives: (1) a lightweight communication protocol based on UNIX sockets for the case when the active backend is co-deployed with the VeloC clients; an RPC-based communication protocol based on Mercury/Thallium that enables the active backend to be deployed on separate nodes. Both alternatives complement the existing default control plane implemented using POSIX shared memory.

In addition, we have added several new features that facilitate better integration with the ECP ecosystem. In addition to continuous integration based on Travis (linked to our public GitHub repository), we designed and developed a test suite for VeloC that verifies the correctness in a multi-node setup on the ECP testbeds using the continuous integration platform provided by ECP (which is configured to mirror the GitHub repository). We are also providing a Spack installation package that is part of the OpenHPC distribution.

We also started several exploratory directions that resulted in several research publications. Notably, we explored how to optimize the checkpoint interval for our multi-level checkpointing strategies used in VeloC. In this context, we devised a technique to reduce the simulation cost of various failure scenarios for a wide range of parameters using machine learning [181]. Furthermore, we also developed specialized checkpointing approaches for deep learning applications. In particular, we explored how to take advantage of specific properties (e.g. multiple model replicas in the case of data-parallel training) in order to reduce the asynchronous checkpointing overhead. Our work DeepFreeze [182] illustrates such techniques based on the idea of augmenting the execution graph with fine-grain tensor copy operations, which then be asynchronously flushed to stable storage. We plan to integrate such approaches with ECP CANDLE, which relies primarily on deep learning.

Next Steps We are working towards several goals: (1) providing a set of C++ client interfaces as well as automated serialization support for common C++ data structures (notably from the STL); (2) investigate and apply mitigation mechanisms for features that are missing or perform suboptimally (notably multiple concurrent MPI instances); (3) continue hardening the integration with existing ECP applications and automated testing infrastructure. In parallel, we will continue to collaborate with the ECP application teams to address new requirements should they arise.

4.4.12 `WBS 2.3.4.14` ECP SZ: Fast, Effective, Parallel Error-bounded Exascale Lossy Compression for Scientific Data

Overview Extreme scale simulations and experiments are generating more data than can be stored, communicated and analyzed. Error-bounded lossy compressor is critical because it can get a very high compression ratio while still respecting data fidelity based on user's requirement on compression errors.

The VeloC-SZ project is extending and improving the SZ error-bounded lossy compressor for structured and unstructured scientific datasets. SZ offers an excellent compression ratio as well as very low distortion and compression time. Further work is essential, however, to improve our SZ lossy compressor for ECP scientific datasets, while ensuring that user-set error controls are respected. Specifically, we are: (i) optimizing SZ compression ratios, accuracy and speed based on end-user needs (ii) refactoring SZ in C++ to support a composable compression framework and all data types used in ECP applications, (iii) integrating SZ in ECP client applications, (iv) developing the GPU version of SZ which supports multiple supercomputers with different architectures (such as Aurora, Frontier and Summit), and (v) improving robustness and testability. We are working with multiple ECP application teams, including ExaSky cosmology teams (HACC), molecular dynamics simulations groups (EXAALT), x-ray laser imaging experimentalists (ExaFEL), and computational chemists (NWChem-X, GAMESS) to optimize SZ for their applications and to harden SZ.

Key Challenges SZ faces several key challenges:

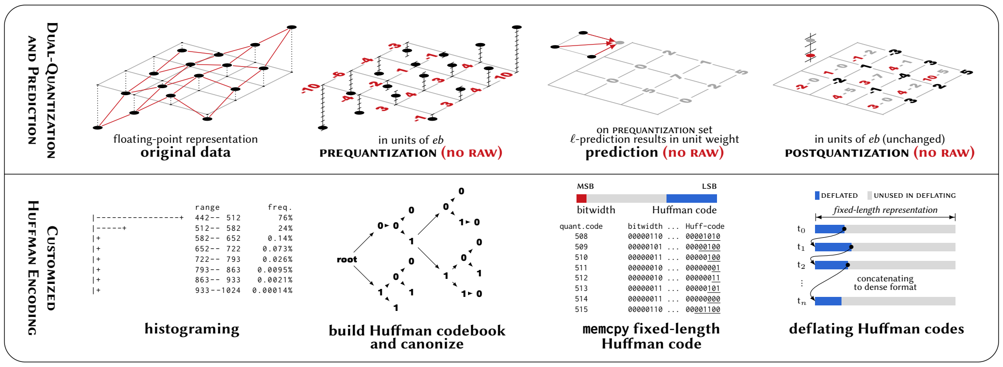
- **Parameter tuning**: One challenge in optimizing lossy compression for scientific applications is the large diversity of scientific data, dimensions, scales, and dynamic data changes in both space and time. Each application requires specific parameters tuning and in some cases, a specific compression pipeline, which is non-trivial to implement.
- **Implementation & optimization over GPU**: A second challenge is the sophisticated design in different stages of the SZ (such as data prediction, Huffman tree construction, Huffman encoding), which makes the development of efficient GPU kernels non-trivial.
- **Diverse integration schemes**: A third challenge is the diversity of the integration schemes for the different ECP client applications: HACC integrates SZ in a proprietary I/O library (GIO), Exafel integrates SZ directly in the LCLS data processing pipeline. GAMESS integrates SZ in the application directly replacing some code sections. NWChem-X integrates SZ for checkpoint/restart.
- **Portable support for GPU**: Optimization of SZ for Aurora and Frontier requires writing portable accelerator codes that are non trivial for complex compression pipeline.
- **Improve development robustness and testability**: The SZ testing infrastructure (unit test, correctness test, performance test, regression test, continuous integration) will need to be adapted and its performance optimized for the new C++ implementation. A template based approach must be used to improve robustness, debugging and testability.

Solution Strategy As for the first challenge, we keep a close communication with ECP application users to understand their specific demands on the lossy compression. For instance, we have a weekly meeting with ECP application teams to discuss the required error bounds, compression speed and quality and integration. We also exploit an adaptive prediction method to optimize the compression quality for diverse datasets.

As for the second challenge, we keep exploring the new strategies to improve the GPU kernel performance for different stages of SZ. This requires in-depth understanding of SZ compression principles, the pros and cons of each step and thorough knowledge about efficient GPU kernel implementation.

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

Concerning the fourth challenge, we are in contact with ALCF and OLCF as well as with vendors to access simulators and early systems that will help to optimize the accelerator implementations. The portable GPU implementation of SZ relies on Kokkos.



Dual-Quantization and Prediction
floating-point representation
original data
in units of eb
prequantization (no raw)
on prequantization set
ℓ-prediction results in unit weight
prediction (no raw)
in units of eb (unchanged)
postquantization (no raw)
Customized Huffman Encoding
histograming
build Huffman codebook and canonize
memcpy fixed-length Huffman code
deflating Huffman codes

Figure 61: cuSZ: Design Overview

Concerning the robustness and testability, we are continuously developing and improving it. We will need to adapt it for C++ as part of the SZ refactoring. We will use ECP testing environment when it becomes available. We also often discuss potential solutions with application teams when needed.

Recent Progress We have created an official website (http://szcompressor.org) to organize different versions/products of SZ (such as CPU version, GPU version). The current flagship products include CPU version of SZ (the classic C version), composable version of SZ (a.k.a., SZ3 in C++), CUDA version of SZ, and Kokkos version of SZ. They are all open source libraries under the BSD3 license.

The CPU version of SZ is the classic state-of-the-art version (a.k.a., SZ 2), whose latest version number is 2.1.10. We keep improving SZ 2 to fit user's diverse requirements from different applications. For example, the new features in 2.1.10 include: the support of user-parameters in python binding particularly for EXAFEL, support for compressing data according to the highest dimension for the specific 3D datasets with high smoothness on the highest dimension; support for fix-value-range; support for printing the statistics related to compression for a better understanding of the compressibility.

SZ 3 breaks down different stages of SZ to form a loosely-coupled model, such that the users can construct a new compressor by customizing each compression step conveniently. By leveraging SZ 3, the new compression method (published in HPDC2020) leverages a more advanced prediction method by combining the second-order Lorenzo and second-order regression. The compression ratio can be improved by 51% with the same data distortion level (PSNR) for the ECP QMCPack application dataset.

We also significantly improved the GPU kernel performance of SZ. We have released the cuda-based SZ (cuSZ) 0.1.1 and tested its performance on V100 GPU (adopted by Summit). The cuSZ is optimized on each compression stage in the SZ framework, including the data prediction, quantization, and Huffman encoding. Specifically, we developed a dual-quantization scheme to entirely remove the data dependency in the prediction step of SZ such that this step can be performed efficiently on GPUs. We developed an efficient customized Huffman coding for the SZ compressor on GPUs. We implemented cuSZ using CUDA and optimized its performance by improving the utilization of GPU memory bandwidth. Figure 61 presents the design overview, and the technical details can be found in our paper published in PACT2020. Experiments on V100 GPU show that the overall compression performance reaches up to 37GB/s based on both HACC and NYX simulation datasets.

We also developed the first prototype of kSZ (short for kokkosSZ), and compared its performance to cuSZ (the CUDA version of SZ) on Summit with CUDA GPUs (i.e., NVIDIA Tesla V100) at OCLF. Kokkos provides a good abstraction for both parallel execution of code and data management, which can be used to support portable implementation across different accelerator technologies. The kSZ exhibits performance very similar (∼1% difference) to cuSZ based in our experiments on Summit.

Next Steps Our next efforts are: (1) Improve compression quality and performance for both cuSZ and kSZ, as well as their stability, (2) keep refactoring SZ in C++ (the first version will be released in Jan, 2021), (3) keep working on the compression quality improvement and integration of SZ in more ECP applications such as EXAALT, and (4) evaluate the portable GPU version of SZ on more ECP platforms.

4.4.13 WBS 2.3.4.15 *ExaHDF5*

Overview Hierarchical Data Format version 5 (HDF5) is the most popular high-level I/O library for scientific applications to write and read data files. The HDF Group released the first version of HDF5 in 1998 and over the past 20 years, it has been used by numerous applications not only in scientific domains but also in finance, space technologies, and many other business and engineering fields. HDF5 is the most used library for performing parallel I/O on existing HPC systems at the DOE supercomputing facilities. NASA gives HDF5 software the highest technology readiness level (TRL 9), which is given to actual systems "flight-proven" through successful mission operations.

In this project, we have developed various HDF5 features are in development to address efficiency and other challenges posed by data management and parallel I/O on exascale architectures. The ExaIO-HDF5 team is productizing features and techniques that have been previously prototyped, exploring optimization strategies on upcoming architectures, maintaining and optimizing existing HDF5 features tailored for ECP applications. Along with supporting and optimizing I/O performance of HDF5 applications, new features in this project include transparent data caching in the multi-level storage hierarchy, topology-aware I/O related data movement in exascale systems, full single-writer / multi-reader (SWMR) for workflows, asynchronous I/O, querying data and metadata, and scalable sub-file I/O.

Many of the funded exascale applications and co-design centers require HDF5 for their I/O, and enhancing the HDF5 software to handle the unique challenges of exascale architectures will play an instrumental role in the success of the ECP. For instance, AMReX, the AMR co-design center, is using HDF5 for I/O, and all the ECP applications that are collaborating with AMReX will benefit from improvements to HDF5. The full SWMR feature will support the needs of ExaFEL's workflow in appending data incrementally. The virtual Object Layer (VOL) and interoperability features with netCDF and ADIOS data open up the rich HDF5 data management interface to science data stored in other file formats. The project will be releasing these new features in HDF5 for broad deployment on HPC systems. Focusing on the challenges of exascale I/O, technologies will be developed based on the massively parallel storage hierarchies that are being built into pre-exascale systems. The enhanced HDF5 software will achieve efficient parallel I/O on exascale systems in ways that will impact a large number of DOE science as well as industrial applications.

Key Challenges

There are challenges in developing I/O strategies for using a hierarchy of storage devices and topology of compute nodes efficiently, developing interoperability features with other file formats, and integrating existing prototyped features into production releases.

Efficient use of hierarchical storage and topology. Data generation (e.g., by simulations) and consumption (such as for analysis) in exascale applications may span various storage and memory tiers, including near-memory NVRAM, SSD-based burst buffers, fast disk, campaign storage, and archival storage. Effective support for caching and prefetching data based on the needs of the application is critical for scalable performance. Also, support for higher bandwidth transfers and lower message latency interconnects in supercomputers is becoming more complex, in terms of both topologies as well as routing policies. I/O libraries need to fully account for this topology in order to maximize I/O performance, and current I/O mechanisms fail to exploit the system topology efficiently.

Asynchronous I/O: Asynchronous I/O allows an application to overlap I/O with other operations. When an application properly combines asynchronous I/O with nonblocking communication to overlap those operations with its calculation, it can fully utilize an entire HPC system, leaving few or no system components idle. Adding asynchronous I/O to an application's existing ability to perform nonblocking communication is a necessary aspect of maximizing the utilization of valuable exascale computing resources.

Solution Strategy *Utilizing complex compute and storage hardware.* To take advantage of multiple levels of faster storage layers between memory and medium- to long-term storage, we developed Data Elevator. The Data Elevator library intercepts HDF5 file access calls and redirects them to intermediate faster caching storage layers, which future application reads or writes will then access. Data Elevator was extensively tested on burst buffers that were shared by all compute nodes. We are currently testing it with node-local burst buffer layer using UnifyFS.

Taking the usage of multiple levels of memory and storage to the next level, we have designed a new virtual object layer (VOL) connector, called “Cache VOL”. With the usage of VOL infrastructure in HDF5, Data Elevator as well as Cache VOL intercept data read and write calls and move the data transparently between source and destination storage levels. As a result, applications can take advantage of these VOL connectors without modifying their source code and avoid placing a burden on users to move the data explicitly to and from intermediate caching storage layers.

Asynchronous I/O Virtual Object Layer (VOL) Connector: Implementation of asynchronous I/O operations can be achieved in different ways. Since the native asynchronous interface offered by most existing operating systems and low-level I/O frameworks (POSIX AIO and MPI-IO) does not include all file operations, we chose to perform I/O operations in a background thread. With the recent increase in the number of available CPU threads per processor, it is now possible to use a thread to execute asynchronous operations from the core that the application is running on without a significant impact on the application’s performance. As shown in Figure 62, when an application enables asynchronous I/O, a background thread is started. Each I/O operation is intercepted, and an asynchronous task is created, storing all the relevant information before inserting it into the asynchronous task queue. The background thread monitors the running state of the application, and only starts executing the accumulated tasks when it detects the application is idle or performing non-I/O operations. When all I/O operations have completed and the application issues the close file call, the asynchronous I/O related resources, as well as the background thread itself, are freed.

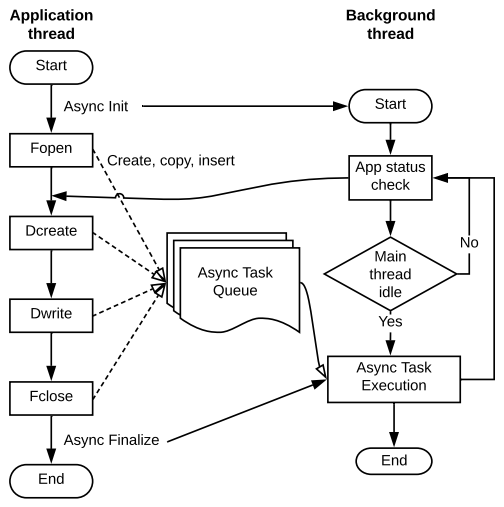


Figure 62: An overview of asynchronous I/O as a HDF5 VOL connector

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

Developed methods to use multi-level storage. The project team has developed a prototype implementation of using memory and storage layers for caching data. The team previously demonstrated that using the burst buffers on Cori, the Data Elevator achieves 1.2–3X performance improvement over a highly tuned HDF5 code in reading data. Performance evaluation included representative I/O of convolution on climate modeling data, gradient computation of a plasma physics data, and vorticity computation of a combustion dataset. The team extended testing Data Elevator to use UnifyFS, a system for managing node-local storage, to take advantage of this storage layer. The team also developed “Cache VOL” connector for take advantage of RAM, node-local cache or any other layer between application buffer and longer-term storage.

Supporting ECP application I/O The ExaIO-HDF5 team has been working with various applications in the ECP portfolio. Applications in the AMReX co-design center have seen some performance issues, mainly because of less optimal configurations, such as using too few file system servers (e.g. Lustre Object Storage Targets or OSTs), producing a large number of metadata requests, using MPI collective buffering that was observing poor performance on NERSC’s Cori. By simply changing these configurations, HDF5 achieved higher performance in writing files. The team also tuned HDF5’s I/O performance by more than 10X by setting the alignment parameter that matches the block size of the GPFS file system on Summit at OLCF.

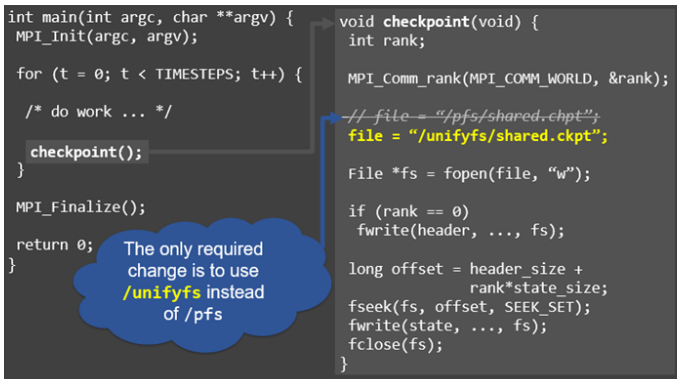
Asynchronous I/O The ExaHDF5 team evaluated the asynchronous I/O VOL connector on Cori at NERSC and on Summit at OLCF with several I/O kernels and ECP application I/O benchmarks. Experimental results show that asynchronous I/O can effectively mask the I/O cost when the application is idle or performing non-I/O operations.

Next Steps The ExaIO-HDF5 team is developing subfiling for reducing locking and contention on parallel file systems, fine-tuning asynchronous I/O and the Cache VOL connectors for caching and prefetching, and supporting ECP AD and ST teams and facilities in improving the overall performance of HDF5.

4.4.14 WBS 2.3.4.15 UnifyCR – A file system for burst buffers

Overview The view of storage systems for HPC is changing rapidly. Traditional, single-target parallel file systems have reached their cost-effective scaling limit. As a result, hierarchical storage systems are being designed and installed for our nation's next-generation leadership class systems. Current designs employ "burst buffers" as a fast cache between compute nodes and the parallel file system for data needed by running jobs and workflows. Burst buffers are implemented as compute-node local storage (e.g., SSD) or as shared intermediate storage (e.g., SSD on shared burst buffer nodes).

Because burst buffers present an additional complexity to effectively using supercomputers, we developed UnifyFS, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. UnifyFS addresses a major usability factor of current and future systems, because it enables applications to gain the performance advantages from distributed burst buffers while providing ease of use similar to that of a parallel file system. To use UnifyFS from within an MPI application, one only needs to change the paths for files that the application uses from the parallel file system to the mount point for UnifyFS, `/unifyfs`. Then the application performs I/O as it normally would, using POSIX I/O or a high level I/O library, e.g., HDF5 or MPI-IO. The UnifyFS library intercepts all I/O operations and manages the file data locally on the compute nodes with high performance.



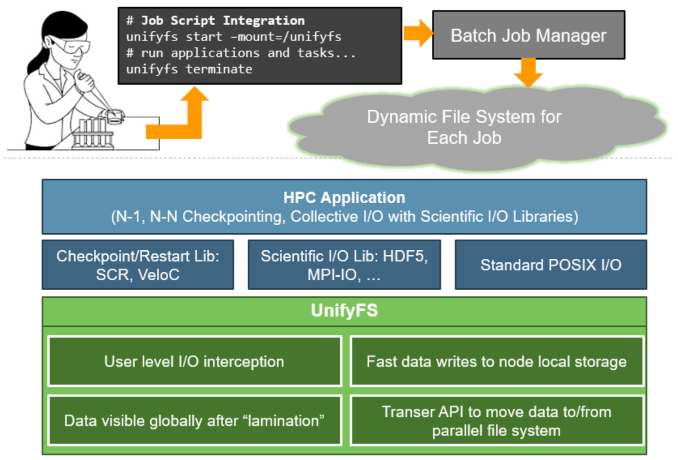
```
int main(int argc, char **argv) {
 MPI_Init(argc, argv);

 for (t = 0; t < TIMESTEPS; t++) {

  /* do work ... */

  checkpoint();
 }

 MPI_Finalize();

 return 0;
}
```

```
void checkpoint(void) {
 int rank;

 MPI_Comm_rank(MPI_COMM_WORLD, &rank);

 ~~// file = "/pfs/shared.chpt";~~
 file = "/unifyfs/shared.ckpt";

 File *fs = fopen(file, "w");

 if (rank == 0)
  fwrite(header, ..., fs);

 long offset = header_size +
               rank*state_size;
 fseek(fs, offset, SEEK_SET);
 fwrite(state, ..., fs);
 fclose(fs);
}
```

Figure 63: Using UnifyFS Using UnifyFS from an MPI application is as easy as using the parallel file system. Simply change the file path to point to the UnifyFS mount point `/unifyfs`, and then perform I/O as normal.

Key Challenges The hierarchical storage of current and future HPC systems includes compute-node local SSDs as burst buffers. This distributed burst buffer design promises fast, scalable I/O performance because burst buffer bandwidth and capacity will automatically scale with the compute resources used by jobs and workflows. However, a major concern for this distributed design is how to present the disjoint storage devices as a single storage location to applications that use shared files. The primary issue is that when concurrent processes on different compute nodes perform I/O operations, e.g., writes, to a shared file, the data for the file are scattered across the separate compute-node local burst buffers instead of being stored in a single location. Consequently, if a process wants to access bytes from the shared file that exist in the burst buffer of a different compute node, that process needs to somehow track or look up the information for locating and retrieving those bytes. Additionally, there is no common interface across vendors for accessing remote burst buffers, so code for cross-node file sharing will not be easily portable across multiple DOE systems with different burst buffer architectures, further increasing programming complexity to support shared files.

For the reasons outlined above, it is clear that without software support for distributed burst buffers, applications will have major difficulties utilizing these resources.



Batch Job Manager
Dynamic File System for Each Job
HPC Application
(N-1, N-N Checkpointing, Collective I/O with Scientific I/O Libraries)
Checkpoint/Restart Lib: SCR, VeloC
Scientific I/O Lib: HDF5, MPI-IO, ...
Standard POSIX I/O
UnifyFS
User level I/O interception
Fast data writes to node local storage
Data visible globally after "lamination"
Transer API to move data to/from parallel file system

```
# Job Script Integration
unifyfs start -mount=/unifyfs
# run applications and tasks...
unifyfs terminate
```

Figure 64: UnifyFS Overview. Users can give commands in their batch scripts to launch UnifyFS within their allocation. UnifyFS works transparently with POSIX I/O, common I/O libraries, and VeloC. Once file operations are intercepted by UnifyFS, they are handled with specialized optimizations to ensure high performance.

Solution Strategy To address this concern, we have developed UnifyFS, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. In Figure 64, we show a high level schematic of how UnifyFS works. Users load UnifyFS into their jobs from their batch scripts. Once UnifyFS is instantiated, user applications can read and write shared files to the mount point just like they would the parallel file system. File operations to the UnifyFS mount point will be intercepted and handled with specialized optimizations that will deliver high I/O performance.

Because bulk-synchronous I/O dominates the I/O traffic most HPC systems, we target our approach at those workloads. Examples of bulk-synchronous I/O include checkpoint/restart and periodic output dumps by applications. Thus, UnifyFS addresses a major usability factor of current and future systems. We designed UnifyFS such that it transparently intercepts I/O calls, so it will integrate cleanly with other software including I/O and checkpoint/restart libraries. Additionally, because UnifyFS is tailored for HPC systems and workloads, it can deliver high performance.

Recent Progress Recently, the UnifyFS team has focused on support for I/O libraries and on read performance. For I/O libraries, we evaluated and improved support for HDF5 and MPI-IO. We rigorously tested UnifyFS with extensive test suites from the two I/O libraries and uncovered and corrected numerous bugs and issues. We also performed a complete overhaul of our metadata infrastructure to improve read performance. With our old infrastructure, read performance was limited by the need to look up data locations in a distributed fashion. Now, most data lookups can be performed locally, resulting in orders of magnitude improvement in read performance. Our source code for UnifyFS is available on GitHub at `https://github.com/LLNL/UnifyFS`.

Next Steps For our next year's effort, we are focused on delivering an implementation of our UnifyFS API that can be used by high level I/O libraries for improved performance. We will also turn our attention to portability to pre-exascale platforms. Additionally, we will continue to improve our support for ECP collaborator software including: I/O libraries HDF5, MPI-IO, PnetCDF, and ADIOS; improving VeloC; and targeting integration with ECP applications such as E3SM, GEOS, and Chombo.

4.4.15 WBS 2.3.4.16 ALPINE

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

ALPINE infrastructure developers come from the ParaView [183, 184] and VisIt [185] teams and ALPINE solutions will deliver in situ DAV functionality in those tools, as well as through Ascent [186], a new in situ infrastructure framework that focuses on flyweight processing. ALPINE focuses on four major activities:

1. Deliver Exascale visualization and analysis algorithms that will be critical for ECP Applications as the dominant analysis paradigm shifts from post hoc (post-processing) to in situ (processing data in a code as it is generated).
2. Deliver an Exascale-capable infrastructure for the development of in situ algorithms and deployment into existing applications, libraries, and tools.
3. Engage with ECP Applications to integrate our algorithms and infrastructure into their software.
4. Engage with ECP Software Technologies to integrate their Exascale software into our infrastructure.

Key Challenges Many high performance simulation codes are using post hoc processing. Given Exascale I/O and storage constraints, in situ processing will be necessary. In situ data analysis and visualization selects, analyzes, reduces, and generates extracts from scientific simulation results during the simulation runs to overcome bandwidth and storage bottlenecks associated with writing out full simulation results to disk. The ALPINE team is addressing two problems related to Exascale processing — (1) delivering infrastructure and (2) delivering performant in situ algorithms. The challenge is that our existing infrastructure tools need to be made Exascale-ready in order to achieve performance within simulation codes' time budgets, support many-core architectures, scale to massive concurrency, and leverage deep memory hierarchies. The challenge for in situ algorithms is to apply in situ processing effectively without a human being in the loop. This means that we must have adaptive approaches to automate saving the correct visualizations and data extracts.

Solution Strategy A major strategy for our team is to leverage existing, successful software, ParaView and its in situ Catalyst [187] library and VisIt, and then to integrate and augment them with ALPINE data and analysis capabilities to address the challenges of Exascale. Both software projects represent long-term DOE investments, and they are the two dominant software packages for large-scale visualization and analysis within the DOE Office of Science (SC) and the DOE National Nuclear Security Agency (NNSA). Our VisIt team is also developing an additional in situ framework, Ascent. Ascent is a "flyweight" solution, meaning that it is focused on a streamlined API, minimal memory footprint, and small binary size. Our solution strategy is two-fold, in response to our two major challenges: infrastructure and algorithms.

For infrastructure, we have developed a layer on top of the VTK-m library for ALPINE algorithms. This layer is where all ALPINE algorithms will be implemented, and it is deployed in ParaView/Catalyst, VisIt, and Ascent. Thus all development effort by ALPINE will be available in all of our tools and by leveraging VTK-m, we will be addressing issues with many-core architectures.

ALPINE is developing a suite of in situ algorithms designed to address I/O and data output constraints and enable scientific discovery. These algorithms include:

Topological analysis can be used to detect features in the data and adaptively steer visualizations with no human in the loop. For example, contour trees can identify the most significant isosurfaces in complex simulations and then the resulting visualizations can use these isosurfaces [188].

Adaptive sampling can be used to guide visualizations and extracts to the most important parts of the simulation, significantly reducing I/O [189, 190, 191].

Statistical feature detection models data using distribution-based approaches and statistical similarity measures to identify and isolate features of interest [192, 193]. Significant data reduction is possible by only saving the statistical representations of the data. Figure 65 illustrates the use of the statistical feature detection approach to identifying bubbles in situ in an MFiX-Exa simulation.

Task-based feature extraction uses segmented merge trees to encode a wide range of threshold based features. An embedded domain specific language (EDSL) can describe algorithms using a task graph abstraction [194, 195] and execute it using different runtimes (e.g., MPI, Charm++, Legion).

Optimal Viewpoint Optimal viewpoint metrics can be used to automate visualization decisions while running in situ. The initial algorithm implementation will choose the best camera placement for a scene, minimizing visualizations written to disk [196, 197].

Lagrangian analysis of vector flow allows more efficient and complete tracking of flow. It can save vector field data with higher accuracy and less storage than the traditional approaches [198, 199, 200].

Recent Progress During this past year, ALPINE has made considerable progress in core functionality in infrastructure and algorithms, and in creating a robust software stack to meet exascale data and viz needs. ALPINE rolled out continuous integration for ParaView and Ascent, improving the Spack build systems. Documentation has been rolled out for algorithms (https://alpine-dav.readthedocs.io/en/latest/) and algorithm teams have added robust unit tests. ALPINE team members have ported ParaView and VisIt to Summit, built Ascent on Iris and are working to port core ALPINE functionality to Tulip. ALPINE is working closely with the VTK-m team to ensure cross-platform portability and the current set of algorithms has been ported to or prototyped in VTK-m.

ParaView utilized the Ascent python extract feature to add in situ visualization in Ascent via ParaView and redesigned the Catalyst in situ library for greater ease of use. As more simulations have adapted in situ approaches, infrastructure integration has shifted from the domain of a small set of VTK-cognizant developers to application scientists who may not have experience with the VTK data model. The new Catalyst adaptor leverages Ascent's Conduit API to describe data and provides schema to convert Conduit mesh descriptions to VTK data objects.

New functionality in Ascent includes Jupyter notebook integration and new derived field quantities, enabling simulations instrumented with Ascent to connect to Jupyter and simulation users to interact with their data in situ using Jupyter Notebooks. Building on its existing query system, Ascent added a production-oriented in situ derived field system leveraging just-in-time (JIT) compilation to target heterogeneous HPC architectures. Ascent's new derived quantities include topological functions, gradients, and basic math functions.

Algorithms have focused on providing new functionality and integration with clients has continued apace. The Ascent $<->$ ExaSky:Nyx integration pipeline has been upgraded to fully run on the GPU. A filter has been developed to allow in situ extraction of visualization slices for Nyx. Two sampling algorithms are now available within Ascent. The initial version uses scalar field values to assign importance to regions of the data while the upgraded algorithm uses both scalar fields and gradient magnitudes.

The Ascent $<->$ PeleC integration has expanded to include ExaLearn. The teams are jointly developing a workflow for tensor-based in situ anomaly detection in high-fidelity combustion Direct Numerical Simulations (DNS). ExaLearn tensor kernels are integrated into the Pele combustion application via Ascent. This workflow has been run on the Summit GPUs for Ascent, ExaLearn, and PeleC and current effort is focusing on the development of specific science use cases.

ALPINE and MFiX-Exa teams are collaborating on bubble finding with the statistical feature detection algorithm and a Catalyst integration. An initial C++ implementation of the statistical feature detection algorithm was integrated directly into the MFiX-Exa simulation to extract bubbles in situ with greater temporal resolution and a factor of 300 in data reduction. A pipeline consisting of the in situ bubble extraction and post hoc Cinema workflow enables interactive exploration of bubble dynamics, Figure 65. The Kitware team recently finished the Catalyst integration into MFiX-Exa while the algorithm team finished an updated VTK-m version. These are currently being merged for testing on Summit's GPU.

The contour tree algorithm recently completed an upgrade and VTK-m port for its distributed parallel version and will circle back to its integration with WarpX. An Ascent $<->$ NekRS integration has been demonstrated on Iris' CPUs and this effort is now looking at science use cases and in situ analysis needs. ALPINE and ExaWind have recently prototyped an Ascent $<->$ AMR:Wind integration and the teams are assessing in situ analysis approaches.

Next Steps Plans for FY21-23 will continue the focus on integration and delivery to ECP applications. The emphasis will shift to porting the team's software products to early hardware and performance studies relevant to ECP applications. The team will also continue outreach and prototyping integrations with ECP application codes in order to facilitate full integration in later years.

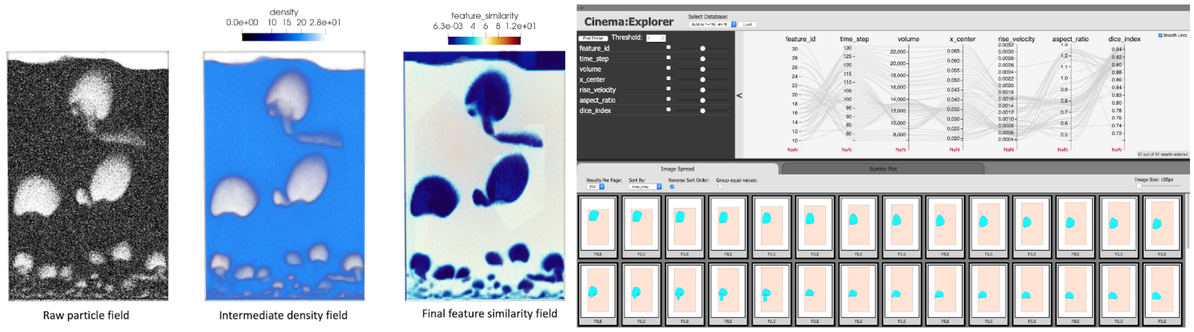


Figure 65: The ALPINE statistical feature detection algorithm is used to identify bubbles in situ in an MFiX-Exa fluidized bed simulation. The raw particle data is converted to a particle density field. A threshold is applied to the density field to create a feature similarity field, separating the bubbles from uninteresting regions. Saving only the statistical representation allows greater temporal resolution while significantly reducing output data size. A preliminary study shows a factor of 300 reduction in data size compared to the raw particle fields. The statistical bubble representation becomes the input to a post hoc Cinema-based workflow to track bubbles and explore bubble dynamics.

4.4.16 WBS 2.3.4.16 ZFP: Compressed Floating-Point Arrays

Overview One of the primary challenges for Exascale computing is overcoming the performance cost of data movement. Far more data is being generated than can reasonably be stored to disk and later analyzed without some form of data reduction. Moreover, with deepening memory hierarchies and dwindling per-core memory bandwidth due to increasing parallelism, even on-node data motion between RAM and registers makes for a significant performance bottleneck and primary source of power consumption.

ZFP is a floating-point array primitive that mitigates this problem using very high-speed, lossy (but optionally error-bounded) compression to significantly reduce data volumes. ZFP reduces I/O time and off-line storage requirements by 1–2 orders of magnitude depending on accuracy requirements, as dictated by user-set error tolerances. Unique among data compressors, ZFP also supports constant-time read/write random access to individual array elements from compressed storage. ZFP's compressed arrays can often replace conventional arrays in existing applications with minimal code changes. This allows the user to store tables of floating-point data in compressed form that otherwise would not fit in memory, either using a desired memory footprint or a prescribed level of accuracy. When used in numerical computations, ZFP arrays provide a fine-grained knob on precision while achieving accuracy comparable to IEEE floating point at half the storage, reducing both memory usage and bandwidth.

This project is extending ZFP to make it more readily usable in an Exascale computing setting by parallelizing it on both the CPU and GPU while ensuring thread safety; by providing bindings for several programming languages (C, C++, Fortran, Python); by adding new functionality, e.g., for unstructured data and spatially adaptive compressed arrays; by hardening the software and adopting best practices for software development; and by integrating ZFP with a variety of ECP applications, I/O libraries, and visualization and data analysis tools.

Key Challenges There are several challenges to overcome on this project with respect to implementing compressed floating-point arrays:

- **Data dependencies.** Compression by its very nature removes redundancies, often by deriving information from what has already been (de)compressed and learned about the data. Such data dependencies can usually be resolved only by traversing the data in sequence, thus complicating random access and parallelism.
- **Random access.** For inline compression, on-demand random access to localized pieces of data is

essential. However, compression usually represents large fixed-length records using variable-length storage, which complicates random access and indexing.

- **Parallelism**. Manycore architectures allow for massively concurrent execution over millions or billions of array elements. Yet compression is usually a process of reducing such multidimensional arrays to a single-dimensional sequence of bits, which requires considerable coordination among parallel threads of execution.
- **Unstructured data**. Unstructured data, such as independent particles and arbitrarily connected nodes in a mesh, has no natural ordering, repeated structure, or regular geometry that can be exploited for compression.
- **Performance**. For inline compression to be useful, both compression and decompression have to be extremely fast (simple), yet effective enough to warrant compression. Moreover, the complexities of compression must be hidden from the user to promote adoption, while allowing sufficient flexibility to support essentially arbitrary data access patterns.

These challenges often suggest conflicting solutions and are further complicated by the extreme demands of Exascale computing applications.

Solution Strategy zfp is unique in supporting read and write random access to multidimensional data, and was designed from the outset to address some of the above challenges. The following strategies are employed on this project to overcome the remaining challenges:

- **Partitioning**. d-dimensional arrays are partitioned into small, independent blocks of 4^d scalars each. This enables both fine-grained random access and a large degree of data parallelism.
- **Fixed-size storage**. Instead of storing fixed-precision values using variable-size storage, zfp uses fixed-size storage to represent values at the greatest precision afforded by a limited bit budget.
- **Adaptive storage**. For applications that demand error tolerances, this project is developing adaptive representations that allocate bits to where they are most needed, which involves efficient management of variable-length records that might expand and shrink in size over time.
- **Parallelism**. OpenMP and CUDA implementations of zfp have been developed that exploit fine-grained data parallelism. Opportunities for task parallelism have also been identified.
- **Preconditioning**. The irregularity and unpredictability of unstructured data is improved using *preconditioners* that "massage" the data to make it more amenable to compression by zfp. Strategies include sorting, binning, structure inference, transposition, pre-transforms like wavelets, etc.
- **Abstraction**. Concrete details about compression, caching, parallelism, thread safety, etc., are abstracted away from the user by providing high-level primitives that make zfp arrays appear like uncompressed arrays, in part via C++ operator overloading. We are designing classes and concepts commonly available for uncompressed arrays, such as proxy references and pointers into compressed storage that act like their uncompressed counterparts; views into and slices of arrays; and iterators compatible with STL algorithms. Such primitives make it easier to write generic code for which zfp arrays may easily be substituted for uncompressed arrays.

Recent Progress Recent work has focused on extending the capabilities of zfp's compressed-array C++ classes. We have refactored these classes to reduce code redundancy between the fixed- and variable-rate representations, added support for 4D arrays, and provided new C and Python language bindings to these classes. This refactoring effort has also decoupled concepts such as array, cache, compression codec, and data store, enabling a more modular design that allows incorporating new codecs and mixed-precision representations under the same array interface. In addition, mutable and immutable proxy references, pointers, and iterators are now available, both for arrays and views into arrays. We have further made progress on parallel decompression of variable-rate streams [201] and released zfp version 0.5.5 [202] as Conda and PIP Python packages. In addition to ST integrations with ADIOS, HDF5, STRUMPACK, and VTK-M, we are integrating ZFP with CEED, EQSIM (Figure 66), QMCPACK, and WARPX.

The results of our R&D efforts have been documented through publications [203, 204, 205, 206, 201, 207], and significant efforts have been made to reach out to customers and the HPC community at large through one-on-one interactions and tutorials, both at ECP meetings and conferences [208, 209, 210, 211, 212, 213]. Together with the SZ team, we will be giving a tutorial and panel at SC20 [214, 215].

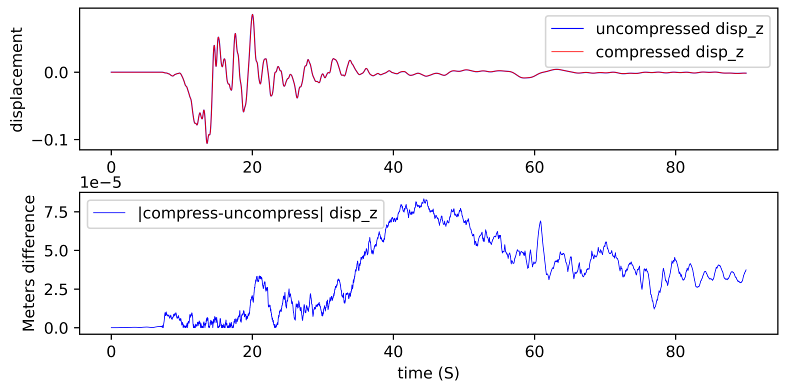


Figure 66: 112:1 ZFP compression of SW4 seismic displacement data to 80μm accuracy.

Next Steps Next year's effort will focus on (1) extending ZFP to support parallel (de)compression of variable-rate streams, and (2) readying ZFP for Frontier by porting our CUDA implementation to HIP. We will also continue our integration efforts with ECP applications and ECP software technologies.

4.5 WBS 2.3.5 SW ECOSYSTEM & DELIVERY

End State: A production-ready software stack delivered to our facilities, vendor partners, and the open source HPC community.

4.5.1 Scope and Requirements

The focus of this effort is on the "last mile" delivery of software that is intended to be supported by DOE Facilities and/or vendor offerings. The scope of this effort breaks down into the following key areas:

- Hardening and broad ST and facility adoption of Spack for easy build of software on all target platforms
- Delivery of formal software releases (Extreme-Scale Scientific Software Stack, or E4S) in multiple packaging formats technologies – from-source builds, modules, and containers
- Oversight of the ST SDKs (Software Development Kits) developed in all five ST L3 areas, with a goal of ensuring the SDKs are deployed as production-quality products at the Facilities, and available to the broader open-source HPC community through coordinated releases
- Development of testing infrastructure (e.g., Continuous Integration) in collaboration with HI 2.4.4 (Software Deployment at the Facilities) for use by ECP teams at the Facilities
- Development and hardening of methods for software deployment through the use of container technology
- Development and hardening of a toolkit of reusable components for scientific workflow management systems
- Informal partnerships with the Linux Foundation's OpenHPC project for potential broader deployment of ST technologies in the OpenHPC ecosystem

A major goal of ST is to ensure that applications can trust that ST products will be available on DOE Exascale systems in a production-quality state, which implies robust testing, documentation, and a clear line of support for each product. This will largely be an integration effort building on both the SDKs project elements defined in each ST L3 area, and tight collaboration and coordination with the Hardware Integration L3 area for Deployment of Software on Facilities (WBS 2.4.4). We will work to develop, prototype, and deliver foundational infrastructure for technologies such as continuous integration and containers in tight collaboration with our DOE facility partners. The ultimate goal is ensuring that the ECP software stack is robustly supported, as well as finding a reach into the broader HPC open-source community – both of which provide the basis for long-term sustainability required by applications, software, Facilities, and vendors who rely upon these products.

Spack has broad adoption in the open-source community as an elegant solution toward solving many of the challenges presented by building software with many dependencies. Spack is one of the most visible outward-facing products in this L3 area, and is the basis for the SDK and E4S efforts.

4.5.2 Assumptions and Feasibility

Success in this effort will require a coordinated effort across the entire hardware and software stack — in particular with HI 2.4.4 (Delivery of Software to Facilities) and in some cases, our vendor partners. This cooperation is a critical first step in enabling our goals, and this area will drive toward ensuring those partnerships can flourish for mutual gain.

Given the project timelines and requirements of production systems at our Facilities, we do not envision a wholly new software stack as a feasible solution. We do however recognize that in many cases the software of today's HPC environments will very likely need to either be evolved or extended to meet the mission goals. This will require first, proof-of-concept on existing pre-Exascale hardware, and ultimately adoption of technologies by system vendors where required, and by other application and software teams.

4.5.3 Objectives

This area will focus on all aspects of integration of the ECP software stack embodied in E4S and the development of SDK Community Policies, and building the workflows community and deploying a toolkit of hardened, reusable components for workflow management systems, with a focus on putting the infrastructure in place (in partnership with HI and the SDKs) for production-quality software delivery through technologies such as Spack, continuous integration, and containers.

4.5.4 Plan

Version 1.2 of the Extreme-Scale Scientific Software Stack (E4S) was released in October, which includes 67 ST products that have Spack packages. This release is downloadable from DockerHub under the ecpe4s area. The E4S Spack Build Cache now includes binaries for ppc64le as well as x86_64 and includes over 22,000 total binaries. The E4S containers now support custom images for ECP applications, such as WDMApp and Pantheon. A regular cadence of E4S releases will continue, with updated ST products, broader facility adoption, and potentially inclusion in vendor offerings.

In close coordination with E4S, a number of SDKs are being developed across the other L3 ST areas, building on the years of experience the xSDK (Math Libraries). These SDKs will become a prime vehicle for our delivery strategy, while also providing ST products with a standard set of community policies aimed at robust production-ready software delivery. In October 2020, Version 1 of the E4S Community Policies was announced. The E4S Community Policies will serve as membership criteria for E4S member packages. The E4S Community Policies will continue to evolve as we work toward release of Version 2 of them.

Spack continues to gain penetration across the ECP, and will be the de facto delivery method for ST products building from source. We provide Spack packaging assistance for ST users and DOE Facilities, and are developing new capabilities for Spack that enable automated deployments of software at Facilities, in containerized environments, and as part of continuous integration. A new effort to support running test suites within Spack environments via the `spack test` functionality is underway and will be integrated with GitLab continuous integration to display dashboards for E4S tests. Concurrently, we are developing technologies and best practices that enable containers to be used effectively at Facilities, and are pushing to accelerate container adoption within ECP.

We filled a gap in the ST portfolio by instantiating a new project on scientific workflows, which is currently in an initial development phase. Within this project, we are establishing the ExaWorks toolkit by assembling shared components from existing workflow projects. The ExaWorks toolkit will provide a robust, well-tested, documented, and scalable set of components that can be combined to enable diverse teams to produce scalable and portable workflows for a wide range of exascale applications. Importantly, the project will not create a new workflow system nor does it aim not to replace the many workflow solutions already deployed and used by scientists, but rather it will provide well engineered and scalable components which can be leveraged by new and existing workflows.

4.5.5 Risks and Mitigation Strategies

- Deploying E4S on unknown architectures – use Spack for deployment to decrease installation complexity
- Keeping updated versions of ST and dependent software in synch after initially achieving SDK interoperability
- Delays in deploying a common CI infrastructure lead to subsequent delays in an integrated software release
- Multiple container technologies in flight will make it hard to come to agreement on a "common" looking solution; may not be possible to generate containers that are both portable and performant
- ECP Application Reliance on workflow management systems that may not be scalable and performant at exascale – mitigate by adopting robust, well-tested, and scalable components
- OpenHPC partnership is ill-defined, and unfunded
- Sustainability of ECP ST capabilities after ECP has ended

4.5.6 Future Trends

Software development kits will gain further traction in their communities as the benefits of interoperability and community policies are demonstrated. We believe these processes will become embedded into the communities and become one of the lasting legacies of ECP and critical for sustainability beyond ECP.

Software deployments will continue to become more complex, especially when we require optimized builds for the unique and complicated exascale architectures. Keeping dependencies updated and the software tested on these systems using continuous integration will tax the resources at the Facilities. Software testing that includes interoperability and scalability tests will require further resources, both in terms of people to write the tests and the hours to regularly run them. These put greater emphasis on using and updating Spack as a solution strategy for large collections of software and tight coordination with HI and Facilities on CI infrastructure and resources.

Containers will become more popular and usable as a way to package the entire environment necessary to run an application on the exascale machines, thereby managing some of the complexity of an application deployment. We expect that performance of an application within a container will be nearly as fast or faster than running the application on bare metal. Application build time will be reduced by using the associated build caches.

Workflows will continue to become more complex to complete their science missions, requiring orchestration of many applications and scripts, executed at various scales across many different resource types, and often reliant on machine learning algorithms for guidance. We expect that hardening workflow management systems and building a community centered around robust and scalable components will be foundational for addressing these complexities. Moreover, we expect that container-based scientific workflows will begin to take off as we transition from demonstrations of applications at scale to performing science with them.

4.5.7 WBS 2.3.5.01 *Software Development Kits*

Overview The ST Software Development Kit (SDK) project supports a set of activities focused on

- establishing Community Policies aimed at increasing the interoperability between and sustainability of ST software packages, using the xSDK [126] community package and installation policies [3] as a model.
- coordinating the delivery of ECP ST products through the Extreme-Scale Scientific Software Stack (E4S) [216], a comprehensive and coherent set of software tools, to all interested stakeholders on behalf of ECP ST, including ECP applications and the broader open source community.

An ECP ST SDK is a collection of related software products (called packages) where coordination across package teams will improve usability and practices and foster community growth among teams that develop similar and complementary capabilities. SDKs have the following attributes:

- Domain scope: Collection makes functional sense.
- Interaction model: How packages interact; compatible, complementary, interoperable.
- Community policies: Value statements; serve as criteria for membership.
- Community interaction: Communication between teams. Bridge culture. Common vocabulary.
- Meta-infrastructure: Encapsulates, invokes build of all packages (Spack), shared test suites.
- Coordinated plans: Inter-package planning. Does not replace autonomous package planning.
- Community outreach: Coordinated, combined tutorials, documentation, best practices.

The SDK project is needed within ECP because it will make it simpler for ECP applications to access required software dependencies on ECP target platforms and drastically lower the cost of exploring the use of additional ECP ST software that may be of benefit. In addition, the SDK effort will decrease the ECP software support burden at the major computing facilities by ensuring the general compatibility of ST packages within a single software environment, providing tool support for the installation of ST packages on Facility machines, communicating common requirements for ST software and facilitating the set up of CI testing at the Facilities. This project works closely with the HI 2.4.4 *Deployment of Software at the Facilities* project.

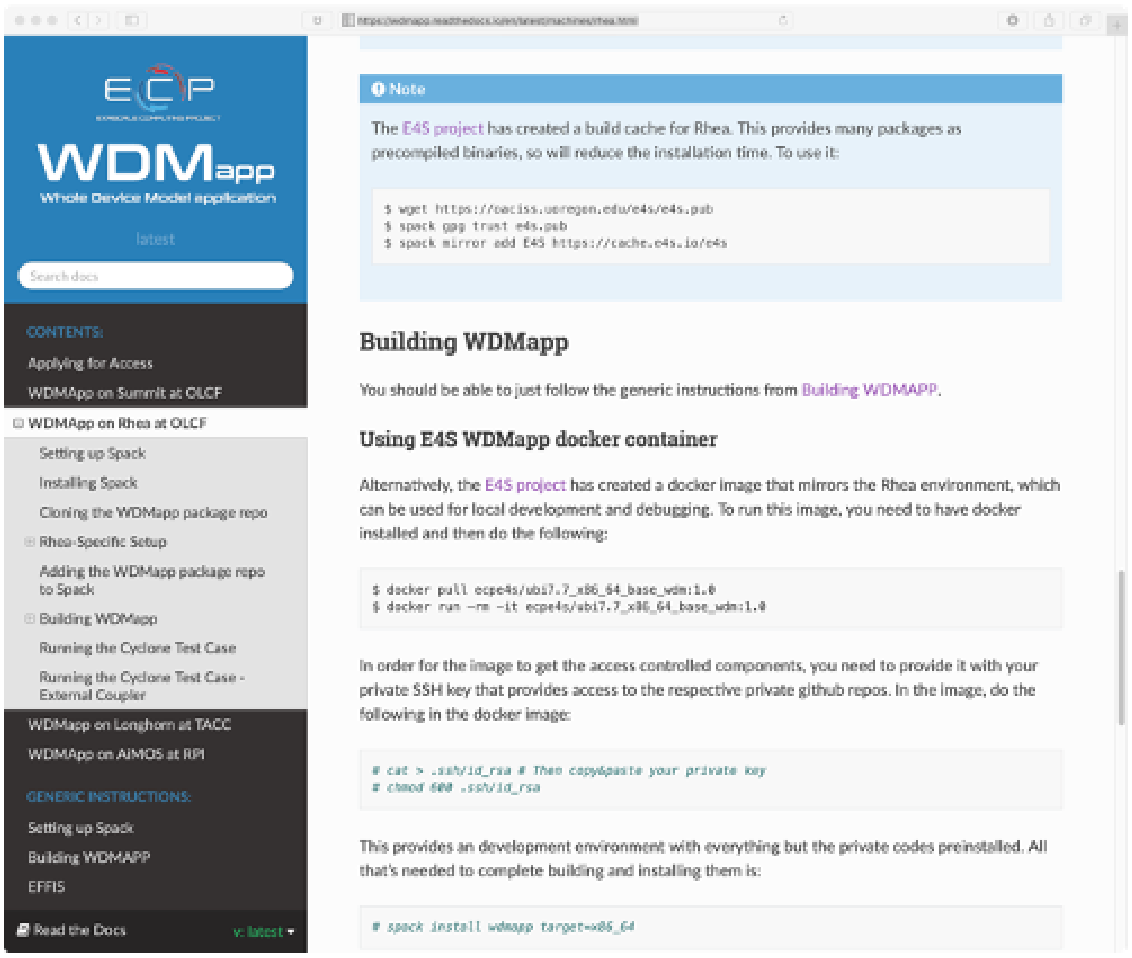
Key Challenges ST software packages have been developed in a variety of very different cultures and are at significantly different levels of software engineering maturity and sophistication. The experience of some of the SDK staff during the formation of the xSDK showed that in this situation, it is challenging to establish common terminology and effective communication, and these are prerequisites to community policies and a robust software release.

Deciding exactly how to deploy the SDKs at the Facilities is itself a challenge. ECP applications will use different combinations of ST software in different configurations. For example, applications will want mathematical libraries capabilities from the xSDK build on top of both MPICH and OpenMPI, and will want different configurations of those mathematical libraries.

Solution Strategy The SDK solution strategy involves pursuing interoperability and sustainability goals by grouping ST software projects into logical collections whose members will benefit from a common set of community policies as well as increased communication between members to standardize approaches where sensible and establish better software practices.

The SDK effort will also facilitate the use of common infrastructure, such as CI testing at the major computing Facilities and the Spack [1] package manager. SDK release and delivery goals will benefit from common package manager and testing infrastructure, including the E4S initiative to provide prebuilt binaries for a variety of architectures.

Recognizing the release readiness and broader maturity differences between ECP ST products, the early release strategy has been to include only those products ready for a joint release in the E4S releases, but to also continue to work with other products in preparing for subsequent release opportunities.



Note

The E4S project has created a build cache for Rhea. This provides many packages as precompiled binaries, so will reduce the installation time. To use it:

```
$ wget https://oaciss.uoregon.edu/e4s/e4s.pub
$ spack gpg trust e4s.pub
$ spack mirror add E4S https://cache.e4s.io/e4s
```

Building WDMapp

You should be able to just follow the generic instructions from Building WDMAPP.

Using E4S WDMapp docker container

Alternatively, the E4S project has created a docker image that mirrors the Rhea environment, which can be used for local development and debugging. To run this image, you need to have docker installed and then do the following:

```
$ docker pull ecpe4s/ubi7.7_x86_64_base_wdm:1.0
$ docker run --rm -it ecpe4s/ubi7.7_x86_64_base_wdm:1.0
```

In order for the image to get the access controlled components, you need to provide it with your private SSH key that provides access to the respective private github repos. In the image, do the following in the docker image:

```
# cat > .ssh/id_rsa # Then copy&paste your private key
# chmod 600 .ssh/id_rsa
```

This provides an development environment with everything but the private codes preinstalled. All that's needed to complete building and installing them is:

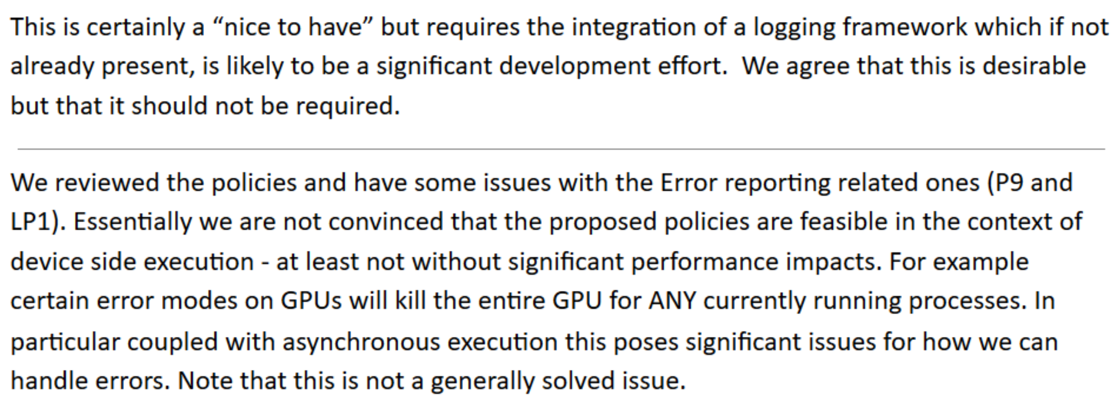
```
# spack install wdmapp target=x86_64
```

Figure 67: WDMapp documentation for how to use the E4S WDMapp Docker container to speed up WDMapp installation by leveraging the E4S Spack build cache.

Recent Progress E4S release 1.0 was announced in November 2019 on the external E4S website [216]. The release supports 50 ST products under Linux x86_64. In February 2020, E4S release 1.1 extended support to both NVIDIA and AMD GPUs with the inclusion of CUDA and ROCm in a single image under Linux x86_64. Release 1.1 also introduced support for the Linux ppc64le platform that supports CUDA 10.1. E4S releases contain HPC as well as AI/ML software including TensorFlow and PyTorch. The E4S DocPortal, accessible from the E4S website, was created to rake information from E4S product GitHub pages and provide it in a single location with the most up-to-date information about releases, installation instructions, etc. The E4S validation testsuite [217] was introduced with support for LLVM and other ST products.

In October 2020, E4S v1.2 included an x86_64 image with 67 E4S products. E4S images are now available for download on DockerHub under the ecpe4s area for testing and will be released on the E4S website in November 2020. The E4S Spack Build Cache now includes binaries for ppc64le as well as x86_64 and includes over 22,000 total binaries. E4S containers now support custom images for ECP applications such as WDMapp and Pantheon (see Figure 67). The E4S build cache has improved the build times for these codes significantly.

Also in October 2020, Version 1 of the E4S Community Policies was announced [8]. The E4S Community Policies will serve as membership criteria for E4S member packages. The E4S Community Policies have their genesis in the xSDK Community Policies, and have a similar purpose. Their purpose is to help address sustainability and interoperability challenges within the complex software ecosystem that ECP ST is a part of.



This is certainly a "nice to have" but requires the integration of a logging framework which if not already present, is likely to be a significant development effort. We agree that this is desirable but that it should not be required.

We reviewed the policies and have some issues with the Error reporting related ones (P9 and LP1). Essentially we are not convinced that the proposed policies are feasible in the context of device side execution - at least not without significant performance impacts. For example certain error modes on GPUs will kill the entire GPU for ANY currently running processes. In particular coupled with asynchronous execution this poses significant issues for how we can handle errors. Note that this is not a generally solved issue.

Figure 68: Two examples of policy feedback received from the ECP ST development community. Comments commonly touched on issues such as appropriateness, both broadly and to specific types of software found within ST, as well as clarity and feasibility.

The process of establishing Version 1 of the E4S Community Policies was a multi-year effort led by the ECP SDK team, including representation from Programming Models and Runtimes, Development Tools, Math Libraries, Data and Vis, and Software Ecosystem and Delivery. This team reviewed the existing xSDK Community Policies and selected those policies that were most generally applicable to all of ECP ST, and not specific to math libraries. From there, the chosen policies were refined and gaps were analyzed.

An early draft was presented to ECP ST leadership, after which an updated draft was socialized with all of ECP ST. The feedback gathered was incorporated into another draft that was again shared broadly across ECP ST. Feedback examples can be seen in Figure 68. After considering the latest feedback, the first version of the policies was finalized. A strong effort was made to involve all interested community members and seriously consider the feedback received.

In addition to the policies shown in Figure 9 a second list of Future Revision policies was also created. These policies are not currently E4S membership criteria, but will be very seriously considered in future versions. In most cases, these policies require further refinement or planning prior to adoption. The topics that these policies address provide information about likely subject areas for E4S policies going forward and are critical to ongoing communication with the E4S community.

Next Steps Current and near-term efforts include:

- Defining a process for documenting and verifying compatibility with E4S Community Policies.
- Assisting with E4S deployment to computing Facilities.
- Adding additional ST software to E4S.
- Assisting with establishing workflows around the maintenance of multi-package CI builds at computing facilities.
- Starting work on Version 2 of the E4S Community Policies.
- Supporting SDK-specific efforts focused on the needs of each SDK, with a particular emphasis on sustainability.

4.5.8 `WBS 2.3.5.09` *Software Packaging Technologies*

Overview ECP is tasked with building the first capable exascale ecosystem, and the foundation of this ecosystem, per ECP's mission statement, is *an integrated software stack*. Building and integrating software for supercomputers is notoriously difficult, and an integration effort for HPC software at this scale is unprecedented. Moreover, the software deployment landscape is changing as containers and supercomputing-capable software package managers like Spack emerge. Spack holds the promise to automate the builds of all ECP software, and to allow it to be distributed in new ways, including as binary packages. Containers will enable entire application deployments to be packaged into reproducible images, and they hold the potential to accelerate development and continuous integration (CI) workflows.

This project will build the tooling required to ensure that packaging technologies can meet the demands of the ECP ecosystem. The project provides Spack packaging assistance for ST users and ECP facilities, and it develops new capabilities for Spack that enable automated deployments of software at ECP facilities, in containerized environments, and as part of continuous integration. Concurrently, the "Supercontainers" sub-project is investigating and developing technologies and best practices that enable containers to be used effectively at ECP facilities. Supercontainers will ensure that HPC container runtimes will be scalable, interoperable, and integrated into Exascale supercomputing across DOE.

Key Challenges Historically, building software to run as fast as possible on HPC machines has been a manual process. Users download source code for packages they wish to install, and they build and tune it manually for high performance machines. Spack has automated much of this process, but it still requires that users *build* software. Spack needs modifications to enable it to understand complex microarchitecture details, ABI constraints, and runtime details of exascale machines. This project will enable binary packaging, and it will develop new technologies that enable the same binary packages to be used within containers *or* in bare metal deployments on exascale hardware.

The Supercontainer effort faces similar challenges to deploying containers on HPC machines. Container technology most notably enables users to define their own software environments, using all the facilities of the containerized host OS. Users can essentially bring *their own* software stack to HPC systems, and they can snapshot an entire application deployment, including dependencies, within a container. Containers also offer the potential for portability between users and machines. The goal of moving an HPC application container from a laptop to a supercomputer with little or no modification is in reach, but there are a number of challenges to overcome before this is possible on Exascale machines. Solutions from industry, such as Docker, assume that containers can be built and run with elevated privileges, that containers are isolated from the host network, filesystem, and GPU hardware, and that binaries within a container are unoptimized and can run on any chip generation from a particular architecture. These go against the multi-user, multi-tenant user environment of most HPC centers, and optimized containers may not be portable across systems.

Solution Strategy The Spack project supports ST teams by developing portable build recipes and additional metadata for the ECP package ecosystem. The end goal is to provide a packaging solution that can deploy on bare metal, in a container, or be *rebuilt* for a new machine on demand. Spack bridges the portability divide with portable package recipes; specialized packages can be built per-site if needed, or lowest-common denominator packages can be built for those cases that do not need highly optimized performance. Packages are relocatable and can be used outside their original build environment. Moreover, Spack provides *environments* that enable a number of software packages to be deployed together either on an HPC system or in a container.

The Supercontainer project seeks to document current practice and to leverage existing container runtimes, but also to develop new enabling technologies where necessary to allow containers to run on HPC machines. Several HPC container runtimes (Shifter, Charliecloud, and Singularity) already exist, and this diversity enables wide exploration of the HPC container design space, and the Supercontainers project will work with their developers to address HPC-specific needs, including container and job launch, resource manager integration, distribution of images at scale, use of system hardware (storage systems, network and MPI libraries, GPUs and other accelerators), and usability concerns around interfacing between the host and container OS (e.g., bind-mounting, etc. required for hardware support).

The project will document best practices and produce a technical report to help educate new users and developers to the advantages of containers, as well as a best-practices report to help ensure efficient container utilization on supercomputers. Both of these will be living documents, periodically updated in response to lessons learned and feedback. In addition, we will identify gaps, and implement changes and automation in one of the three existing runtimes, as needed. The project will also interface with the E4S and SDK teams, as well as AD teams interested in containerizing their applications. We will work to enable these teams to deploy reproducible, minimally-sized container images that support multiple AD software ecosystems.

Recent Progress

1. Released `archspec`[6], a library for labeling and distributing CPU microarchitectures, as a spin-off project from Spack.
2. Developed the `spack containerize` command that enables users to easily build lightweight containers from Spack environments.
3. Worked with the E4S team to get E4S packages building on the Ascent machine at OLCF, on NERSC's Cori machine, and in continuous integration in Amazon Web Services (on optimized `skylake` instances).
4. Continued to support ECP ST teams and conducted a survey to better understand their usage of Spack.
5. Developed optimized E4S container images for `ppc64le`, `x86_64`, as well as CUDA and ROCm GPUs. Solutions for heterogeneous hardware usage via containers are ahead of available solutions in industry. [7].
6. Continued to test multiple container runtimes on pre-exascale systems.

Next Steps

1. Implement support for running test suites within Spack environments, and develop ways for teams to build portable tests associated with Spack packages ("`spack test`").
2. Integrate `spack test` with GitLab continuous integration to display dashboards for E4S tests run on EA systems.
3. Work with the E4S team to implement initial package smoke tests for E4S on EA systems.
4. Improve automation on GitHub by implementing a change notification bot for Spack package maintainers, to tell them when others have proposed changes to their packages.
5. Implement pull request testing for Spack GitHub, so that every pull request to Spack has its package builds tested in a sandboxed environment to ensure that the `develop` branch continues to build.
6. Investigate unprivileged container builds at ECP facilities and determine which runtimes provide the best support for this.
7. Continue container training sessions, outreach efforts, and tutorials.

[6]https://github.com/archspec/archspec/
[7]https://hub.docker.com/u/ecpe4s

4.5.9 WBS 2.3.5.10 ExaWorks

Overview Exascale computing capacity reinforces the need for workflows and also creates a slew of new workflow challenges. Most notably, the increasing scale and hardware heterogeneity demands higher level programming environments, such as workflows, to enable a broad range of scientists, students, and developers to describe complex computational procedures and manage their execution at enormous scales in intuitive and productive ways. Further, in addition to the changing system architectures, application patterns are also changing: no longer is research conducted with a single invocation of a lone executable, instead it typically requires orchestration of many applications and scripts, executed at various scales across many different resource types, and often reliant on machine learning algorithms for guidance. As a result, exascale Workflow Management Systems (WMSs) will need to support high performance execution of significant numbers of short-duration tasks (e.g., inference tasks), efficient scheduling of tasks with varying resource (e.g., single core, multiple nodes, and accelerators) and time-sensitive (e.g., coupling data analysis with simulations) constraints, and flexible coordination and communication patterns between many concurrent jobs and/or tasks.

Key Challenges Emerging exascale workflows pose significant challenges to the creation of portable, repeatable, and performant workflows. These challenges are both technical and non-technical. On the technical side, WMSs are currently incapable of supporting the needs of heterogeneous co-scheduled and high-throughput workflows, as well as enabling communication between fine grain tasks in dynamic workflows. On the non-technical side, the myriad WMSs that exist, lack of reusable WMS components, and the lack of clear user guidance when selecting a WMS has resulted in a disjoint workflows community that tends toward building ad hoc or bespoke solutions rather than adopting and extending existing solutions.

Specific challenges include:

1. Workflows community: the workflows, applications, and facility communities are disjoint. Efforts are needed to bring these groups together to agree on common workflow components and interfaces, and to work together to develop, integrate, and support these components.
2. Scheduling: exascale workflows must manage the efficient execution of diverse tasks (e.g., in runtime, resource requirements, single/multi-node) with complex interdependencies on increasingly heterogeneous resources.
3. Scale and performance: emerging workflows feature huge ensembles of short-running jobs, which can create millions or even billions of tasks that need to be rapidly scheduled and executed.
4. Coordination and communication: workflows depend on coordination between the workflow and the tasks within the workflow, a need that requires efficient exchange of data following various communication patterns.
5. Fault tolerance: the enormous number of computing elements and workflow tasks increases the likelihood of encountering faults within a workflow both at the system level and also from the millions of concurrent tasks.
6. Portability: most WMSs are tested on a handful of systems and the frequency by which system hardware and software change makes it impossible to guarantee that a workflow will work on even the same system in the future.

Solution Strategy The ExaWorks project will lay the foundation for an inherently *new approach* to workflows: establishing the ExaWorks toolkit (see Figure 69) by assembling shared components from existing workflow projects. The ExaWorks toolkit will provide a robust, well-tested, documented, and scalable set of components that can be combined to enable diverse teams to produce scalable and portable workflows for a wide range of exascale applications. Importantly, the project will not create a new workflow system nor does it aim not to replace the many workflow solutions already deployed and used by scientists, but rather it will provide well engineered and scalable components which can be leveraged by new and existing workflows.

The goals of the initial phase of the project are to instantiate the ExaWorks community, bringing together workflow tool developers, ECP applications, and DOE compute facility representatives. Specifically, it will:

1. Engage the facilities to survey the state of workflow tools and capabilities and ways in which ExaWorks can enhance their capabilities;
2. Establish an advisory board composed of representatives of DOE compute facilities, ECP applications, and workflow tools, to guide and advise ExaWorks;
3. Survey ECP applications teams to identify the tools currently being used and to identify common challenges and needs;
4. Assemble a functional design working group to develop a community-centered draft function design; and
5. Collaborate with ECP applications to develop a proof-of-concept integration using a shared functional component as defined by the draft design, in an ECP application.

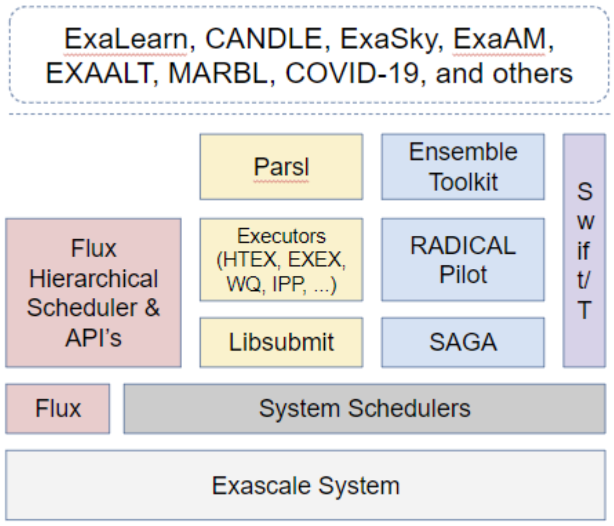

ExaLearn, CANDLE, ExaSky, ExaAM, EXAALT, MARBL, COVID-19, and others
Parsl
Ensemble Toolkit
Swift/T
Flux Hierarchical Scheduler & API's
Executors (HTEX, EXEX, WQ, IPP, ...)
RADICAL Pilot
Libsubmit
SAGA
Flux
System Schedulers
Exascale System

Figure 69: ExaWorks Toolkit

Recent Progress In this first period of the project we have assembled our advisory committee, started a functional design working group, and distributed a workflows survey to the ECP community. The results of this survey are helping to prioritize in-person interviews as well as informing the functional design process and helping to identify initial ExaWorks components. Our team have started prototyping efforts to explore component-based approaches in existing workflow systems. Specifically, we have developed prototype Balsam and RADICAL-Pilot executors for Parsl which enable Parsl workflows to leverage the resource management capabilities of these external systems.

Next Steps The remainder of this initial effort focuses on four important areas. First, continuing to grow the ExaWorks community by engaging with ECP applications, facilities, and WMS teams. Second, working with these partners and stakeholders to produce a draft functional design document that outlines ExaWorks components and potential interfaces to these components. Third, we will produce a report, derived from interviews from the broad ECP community that outlines ECP workflows needs, challenges, and potential solutions. Finally, we will demonstrate the technical feasibility of the ExaWorks approach via application of preliminary components to at least one ECP application.

4.6 WBS 2.3.6 NNSA ST

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

4.6.1 Scope and Requirements

The NNSA ST L3 area was created in FY20, although the projects included have all been part of the ECP before its creation. The capabilities of these software products remains aligned with the other Software Technology L3 areas from which they were derived, but are managed separately for non-technical reasons out of scope of this document.

The resulting products in this L3 area are open source, important or critical to the success of the NNSA National Security Applications, and are used (or potentially used) in the broader ECP community. The products in this L3 span the scope of the rest of ST (Programming Models and Runtimes, Development Tools, Math Libraries, Data Analysis and Vis, and Software Ecosystem), and will be coordinated with those other L3 technical areas through a combination of existing relationships and cross-cutting efforts such as the ST SDKs and E4S.

4.6.2 Objectives

The objective of these software products is to support the development of new from-scratch applications within the NNSA that were started just prior to the founding of the ECP under the ATDM (Advanced Technology Development and Mitigation) program element within NNSA and ASC. While earlier incarnations of these products may have been more research-focused, by the time of the ECP ST restructuring in 2019 that resulted in this L3 area, these products are in regular use by their ATDM applications, and have matured to the point where they are ready for use within the broader open source community.

4.6.3 Plan

NNSA ST products are developed with and alongside a broader portfolio of ASC products in an integrated program, and are planned out at high level in the annual ASC Implementation Plan, and in detail using approved processes within the home institution/laboratory. They are scoped to have resources sufficient for the success of the NNSA mission, as well as a modicum of community support (e.g., maintaining on GitHub, or answering occasional questions from the community).

For ECP products not part of the NNSA portfolio that have critical dependencies on these products, there are often other projects within ECP that provide additional funding and scope for those activities. In those cases, there may be additional information within this document on these products.

4.6.4 Risks and Mitigation Strategies

One risk associated with the NNSA ST projects is the programming environment of the El Capitan system. The programming environment on this system will be a departure from what the NNSA software teams have used before, so there is a risk that it will present challenges that cause delays in porting the software to the system. That said, the probability of this risk is low because the programming environment of El Capitan will also be installed on DOE predecessor machines so it is likely that it challenges will be identified and addressed by the time of El Capitan. NNSA ST projects can mitigate this risk by evaluating their software on the predecessor systems to identify challenges early.

Another risk associated with the NNSA ST L3 is that the projects rely on multiple sources of funding outside of ECP. The budget priorities of those external sources may not always be aligned with those of ECP. In general, this risk is low because the L4 leads strive to align their project goals across all funding sources. However, it is possible that funding expected to be leveraged to develop a feature to later be used for ECP purposes may be dropped. If this occurs, the L4 leads will need to mitigate the situation according to their individual project needs, perhaps by renegotiating deliverable time lines.

Another risk is that others in the community will pick up these products as open source, and expect additional support beyond the scope of the primary NNSA mission. If those dependent products are within the ECP, the main mitigation is to use ASCR contingency funding to provide additional development and support - potentially through support of teams outside of the home institution. If those dependent products are in the broader community, mitigations are generally outside of the scope of the ECP - although each NNSA lab typically has some sort of project (or possibly even a policy) on how to deal with external demands on open source products.

4.6.5 WBS 2.3.6.01 *LANL ATDM Software Technologies*

Overview

The LANL ATDM PMR effort is focusing on the development and use of advanced programming models for Advanced Technology Development and Mitigation (ATDM) use-cases. Our current focus is on research and development of new programming model capabilities in the **Legion** data-centric programming system. Legion provides unique capabilities that align well with our focus on the development of tools and technologies that enables a separation of concerns of computational physicists and computer scientists. Within the ATDM PMR effort we have focused on the development of significant new capabilities within the Legion runtime that are specifically required to support LANL's ATDM applications. Another key component of our work is the co-design and integration of advanced programming model research and development within **FleCSI**, a Flexible Computational Science Infrastructure. A major benefit to the broader ECP community is the development of new features in the Legion programming system which are available as free open-source software `https://gitlab.com/StanfordLegion/legion`.

The **Kitsune** Project, provides a compiler-focused infrastructure for improving various aspects of the exascale programming environment. At present, efforts are primarily focused on advanced LLVM compiler and tool infrastructure to support the use of a *parallel-aware* intermediate representation. In addition, we are actively involved in the Flang Fortran front-end that is now an official sub-project within the overall LLVM infrastructure. All these efforts include interactions across ECP as well as with the broader LLVM community and industry.

The LANL ATDM **Cinema** project develops scalable solutions for data analysis as part of the Data and Visualization software stack. Cinema is a novel database approach to saving data extracts in situ which are then available for post hoc interactive exploration. These data extracts can include metadata, parameters, data visualizations, small meshes, output plots, etc. Cinema workflows enable flexible data analysis using a fraction of the file storage. Cinema ECP workflows that integrate applications, in situ and post-processing analysis are captured and curated through the Pantheon project, which is focused on reproducible ECP workflows. By integrating E4S caches of both applications and dependent capabilities (Ascent, etc.), Pantheon workflows can be downloaded, built and run quickly enough to be useful in a variety of applications such as CI, prototyping functionality or testing analyses.

The **BEE/Charliecloud** subproject is creating software tools to increase portability and reproducibility of scientific applications on high performance and cloud computing platforms. Charliecloud [218] is an unprivileged Linux container runtime. It allows developers to use the industry-standard Docker [219] toolchain to containerize scientific applications and then execute them on unmodified DOE facility computing resources without paying any performance penalty. BEE [220] (Build and Execution Environment) is a toolkit providing users with the ability to execute application workflows across a diverse set of hardware and runtime environments. Using Bee's tools, users can build and launch applications on HPC clusters and public and private clouds, in containers or in containers inside of virtual machines, using a variety of container runtimes such as Charliecloud and Docker.

Key Challenges

Legion: Applications will face significant challenges in realizing sustained performance on next-generation systems. Increasing system complexity coupled with increasing scale will require significant changes to our current programming model approaches. This is of particular importance for large-scale multi-physics applications where the application itself is often highly dynamic and can exhibit high variability in resource utilization and system bottlenecks depending on what physics are currently in use (or emphasized). Our goal in the LANL ATDM PMR project is to support these highly dynamic applications on Exascale systems, providing improvements in productivity, long-term maintainability, and performance portability of our next-generation applications.

FleCSI Legion integration: FleCSI is a Flexible Computational Science Infrastructure whose goal is to provide a common framework for application development for LANL's next-generation codes. FleCSI is required to support a variety of different distributed data structures and computation on these data structures including structured and unstructured mesh as well as mesh-free methods. Our work in the LANL ATDM PMR project is focused on co-designing the FleCSI data and execution model with the Legion programming model to ensure the latest advancements in the programming model and runtimes research community are represented in our computational infrastructure. A significant challenge in our work is the additional constraint that FleCSI must also support other runtime systems such as MPI. Given this constraint, we have chosen an approach that ensures functional correctness across both runtimes but that also leverages and benefits from capabilities in Legion that are not directly supported in MPI (such as task-based parallelism as a first-class construct).

Kitsune: A key challenge to our efforts is reaching agreement within the broader community that a parallel intermediate representation is beneficial and needed within LLVM. This not only requires showing the benefits but also providing a full implementation for evaluation and feedback from the community. In addition, significant new compiler capabilities represent a considerable effort to implement and involve many complexities and technical challenges. These efforts and the process of up-streaming potential design and implementation changes do involve some amount of time and associated risk.

Additional challenges come from a range of complex issues surrounding target architectures for exascale systems. Our use of the LLVM infrastructure helps reduce many challenges here since many processor vendors and system providers now leverage and use LLVM for their commercial compilers.

Cinema Interfacing to a large number of ECP applications with the Cinema software and the management of the voluminous data from these applications.

Bee/CharlieCloud Other HPC-focused container runtimes exist, such as NERSC's Shifter [221] and Singularity [222]. These alternative runtimes have characteristics, such as complex setup requirements and privileged user actions, that are undesirable in many environments. Nevertheless, they represent a sizable fraction of the existing HPC container runtime mindshare. A key challenge for BEE is maintaining support for multiple runtimes and the various options that they require for execution. This is especially true in the case of Singularity, which evolves rapidly. Similarly, there is a diverse collection of resources that BEE and Charliecloud must support to serve the ECP audience. From multiple HPC hardware architectures and HPC accelerators such as GPUs and FPGAs, to differing HPC runtime environments and resource managers, to a multitude of public and private cloud providers, there is a large set of available resources that BEE and Charliecloud must take into consideration to provide a comprehensive solution.

Solution Strategy

Legion: In funded collaboration with NVIDIA, LANL and NVIDIA are developing new features in Legion to support our applications. Necessary features are identified through direct engagement with application developers and through rapid development, evaluation, and refactoring within the team. Major features include Dynamic Control Replication for improved scalability and productivity and Dynamic Tracing to reduce runtime overheads for applications with semi-regular data dependencies such as applications with stencil-based communication patterns.

FleCSI Legion integration: LANL staff work on co-design and integration of the Legion programming system into the FleCSI framework. We have regular milestones that align well with application needs and the development of new features within Legion.

Kitsune: Given the project challenges, our approach takes aspects of today's node-level programming systems (e.g. Kokkos) and programming languages (e.g. C++) into consideration and aims to improve and expand upon their capabilities to address the needs of ECP and LANL's mission critical applications. This allows us to attempt to strike a balance between incremental improvements to existing infrastructure

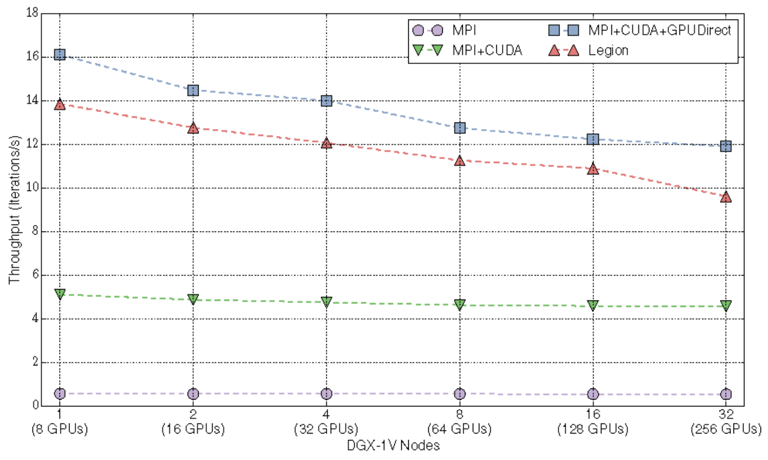


Figure 70: Productivity features such as Dynamic Control Replication scales well across multi-GPU systems in unstructured mesh computations.

and more aggressive techniques that seek to provide innovative solutions, thereby managing risk while also providing the ability to introduce new technologies via the toolchain.

Unlike current designs, our approach introduces the notion of explicit parallel constructs into the LLVM intermediate representation, building off of work done at MIT on Tapir [223] and the OpenCILK effort (urlhttps://cilk.mit.edu). We are working with MIT to extend this work as well as making some changes to fundamental data structures within the LLVM infrastructure to assist with and improve analysis and optimization passes.

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

Cinema [224] is being developed in coordination with LANL's ECP application NGC to ensure that data collected during the simulation execution is of appropriate frequency, resolution, and viewport for later analysis and visualization by scientists. Cinema is an innovative way of capturing, storing and exploring extreme scale scientific data. Cinema is essential for ECP because it embodies approaches to maximize insight from extreme-scale simulation results while minimizing data footprint

Bee/CharlieCloud: The BEE/Charliecloud project is focusing first on providing support for containerized production LANL scientific applications across all of the existing LANL production HPC systems. The BEE/Charliecloud components required for production use at LANL will be documented, released and fully supported. Follow-on development will focus on expanding support to additional DOE platforms. This will mean supporting multiple hardware architectures, operating systems, resource managers, and storage subsystems. Support for alternative container runtimes, such as Docker, Shifter, and Singularity is planned.

Recent Progress

Legion: One of the strengths of Legion is that it executes asynchronous tasks as if they were executed in the sequence they occur in the program. This provides the programmer with a mental model of the computation that is easy to reason about. However, the top-level task in this tree-of-tasks model can often become a sequential bottleneck, as it is responsible for the initial distribution of many subtasks across large machines. In earlier work NVIDIA developed the initial implementation of control replication, which allows the programmer to write tasks with sequential semantics that can be transparently replicated many times, as directed by the Legion mapper interface, and run in a scalable manner across many nodes. Dynamic control replication is an important capability for LANL's ATDM effort, allowing our application teams to write

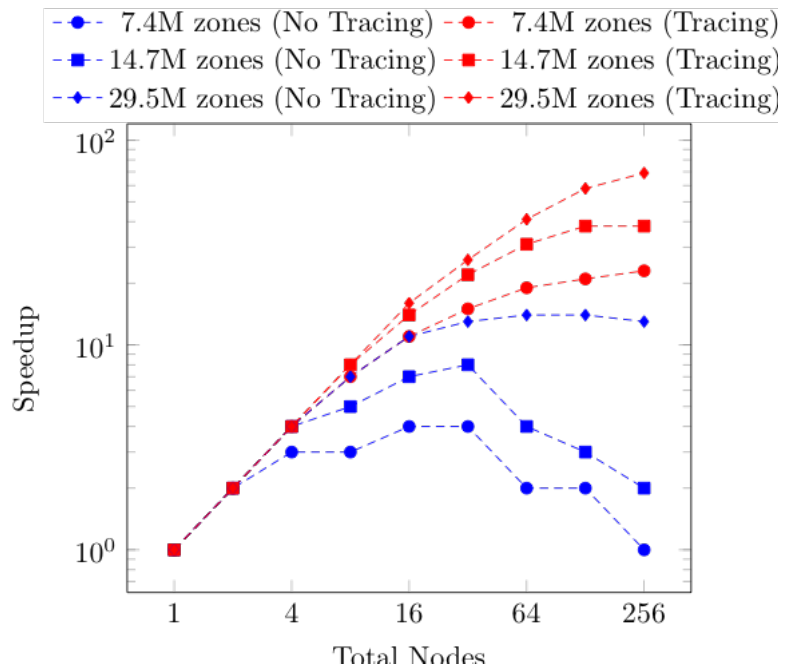


Figure 71: New Legion features such as Tracing will improve strong scaling in unstructured mesh computations.

applications with apparently sequential semantics while enabling scalability to Exascale architectures. This approach will improve understandability of application code, productivity, and composability of software and ease the burden of optimization and porting to new architectures. New dynamic tracing ability has been added to Legion to allow debugging and insight in to performance optimization activities.

FleCSI Legion Integration: A key component of LANL's Advanced Technology Development and Mitigation effort is the development of a flexible computational science infrastructure (FleCSI) to support a breadth of application use cases for our Next Generation Code. FleCSI has been co-designed with the Legion programming system in order to enable our Next Generation Code to be performance portable and scalable to future Exascale systems. Legion provides the underlying distributed and node-level runtime environment required for FleCSI to leverage task and data parallelism, data dependent execution, and runtime analysis of task dependencies to expose parallelism. We completed testing of Legion on Sierra with a Visco-Plastic Self-Consistent, VSCP, application to investigate initial performance on GPU systesm.

Kitsune: Our primary focus is the delivery of capabilities for LANL's ATDM Ristra application (AD 2.2.5.01). In support of the requirements for Ristra, we are targeting the lowering of *"forall"* constructs, including Kokkos `parallel_for` construct, directly into the parallel representation. At present, this works for many C++ constructs (e.g., for and for-range statements). We can target this code to different runtimes and architectures via the compiler and thus avoid reimplementation of Kokkos or fundamental C++ constructs. In addition we are working to replace LLVM's dominator tree, a key data structure for optimizations including parallelization and memory usage analysis, with a *dominator directed-acyclic-graph* (DAG). This capability is still in its early evaluation state and we continue to explore correctness and compatibility within the overall LLVM infrastructure. We are actively watching recent events within the LLVM community around multi-level intermediate representations (MLIR) and the relationship they have with parallel semantics, analysis, optimization, and code generation.

At present we are successfully compiling our full applications using the new toolchain. We continue to test, debug, and work towards improved optimizations and performance.

Cinema: Recent Cinema progress has focused on development of exascale workflows, development of python-based Cinema functionality, and supporting Cinema export capabilities through ALPINE's exascale-capable infrastructure. ParaView's v5.9 release includes a significant rewrite to create *extract generators* to output images and other extracts. The creation of Cinema databases is part of the extract generator workflow both in post hoc ParaView usage or via the in situ Catalyst library. Cinema export is also available via ALPINE's Ascent infrastructure and through VisIt. Cinema capabilities provide scientists more options in analyzing and exploring the results of large simulations by providing a workflow that 1) detects features in situ, 2) captures data artifacts in Cinema databases, 3) promotes post-hoc analysis of the data, and 4) provides data viewers that allow interactive, structured exploration of the resulting artifacts. In our milestones during FY20, we extended two end-to-end reproducible simulation pipelines with ECP applications at scale to generate Cinema databases and ran Cinema-based workflows with Cinema algorithms to produce secondary set of artifacts. We ran (1) Nyx integrated with Ascent, running an ALPINE adaptive sampling algorithm; and (2) SW4 integrated with Ascent, running a VTK-m isocontour algorithm. We ran scaling and performance testing for typical ECP-based Cinema use cases. Lastly we did the annual release of the Cinema toolkit. Based on user feedback, we changed the toolkit from a set of viewers and a command line tool to a single Python module that includes current Cinema toolkit components such as viewers, database classes, preliminary Composable Image Sets classes, Jupyter notebook classes, and a small web server to provide a new way to view databases with the existing viewers. The Cinema Python module is also included in ParaView v5.9. An example of a Jupyter notebook-based approach is shown in Figure 72 with an ExaSky:Nyx volume displayed within a notebook workflow. The Cinema team is working with Exascale science applications to develop in situ and post hoc workflows based on data extracts such as in Figure 65 where bubbles detected in situ are then analyzed within a Cinema viewer to enable studies of bubble dynamics.

Bee/CharlieCloud Recent Charliecloud progress has focused on understanding and documenting best practices for running large scale MPI jobs using containerized runtimes. Charliecloud is enhancing support for multiple MPI implementations. Charliecloud is available at https://github.com/hpc/charliecloud and is distributed inside of Debian and Gentoo Linux distributions as well as being part of OpenHPC. Charliecloud won an 2018 R&D-100 award.

BEE fully supports launching Charliecloud containers on all LANL HPC systems. It can also launch containers on AWS and OpenStack clouds such as NSF Chameleon. BEE also supports interactive launching of jobs with the SLURM resource manager. BEE was shown at the end of FY19 to support a complex multiphysics application with setup, in situ visualization and checkpoint-restart on a production system at LANL.

Next Steps

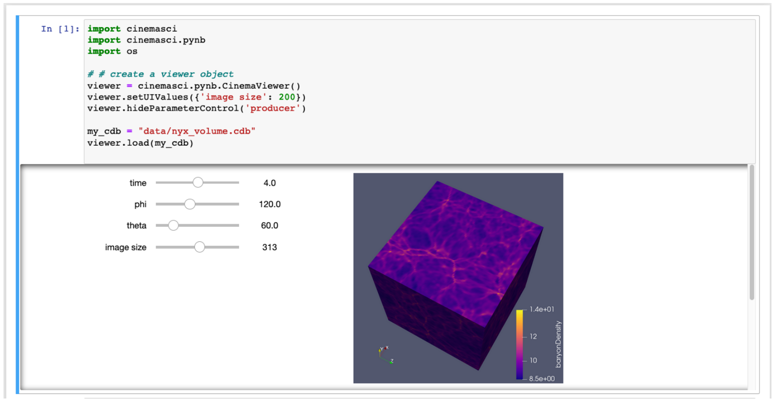
Legion: Focus on hardening and scalability of Legion's Dynamic Control Replication and development of Dynamic Tracing for application use-cases.

FleCSI Legion Integration: Support the Ristra Application milestone to run on Sierra and Trinity.

Kitsune: The key next steps for our efforts are to expand our test cases by increasing the complexity of the codes we're compiling, supporting additional forms of Kokkos constructs, and support other parallel constructs that meet the needs of Ristra. Where possible, we will explore a broader set of use cases within the ECP community. This will be done while also striving to maintain a feature set in Kitsune that matches the most recent releases of the LLVM infrastructure. We will continue a quarterly release cycle of the software and also when feature sets align with our milestones. This work will go hand-in-hand with the code generation and optimization for the exascale system target processors: including CPUs and GPUs on the target platforms. The development of associated runtime targets that can reduce code generation complexity will also be a component of our future efforts (as needed).

With the addition to Fortran to LLVM via the Flang front end we will also look to add support for lowering to the parallel IR in those use cases.

Cinema: In FY21, Cinema will be hardening the Composable Image Sets format to meet user requests and adding functionality to the Python-based components, demonstrating these capabilities with ATDM/ECP data and applications. Cinema is focusing on outreach to ECP applications to identify new application workflows that can be reasonably made efficient and working on new analysis methods for Cinema users.



```
In [1]: import cinemasci
        import cinemasci.pynb
        import os

        # # create a viewer object
        viewer = cinemasci.pynb.CinemaViewer()
        viewer.setUIValues({'image size': 200})
        viewer.hideParameterControl('producer')

        my_cdb = "data/nyx_volume.cdb"
        viewer.load(my_cdb)
```

```
      time   4.0
       phi   120.0
     theta   60.0
image size   313
```

Figure 72: A screenshot of an ECP Nyx simulation in the new jupyter notebook-based cinemasci module.

Bee/CharlieCloud A refactoring of BEE to support an open standard is underway. Support for the Open Workflow standard will allow a base on a well defined workflow description language leveraged by other scientific communities. This will then be tested on multiple systems to ensure portability.

4.6.6 WBS 2.3.6.02 LLNL ATDM Software Technologies

Overview

Spack is a package manager for HPC [225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 2, 236, 237, 238, 239]. It automates the process of downloading, building, and installing different versions of HPC applications, libraries, and their dependencies. Facilities can manage multi-user software deployments, and developers and users can manage their own stacks separately. Spack enables complex applications to be assembled from components, lowers barriers to reuse, and allows builds to be reproduced easily.

The **MFEM** library [240, 241] is focused on providing high-performance mathematical algorithms and finite element discretizations to next-gen high-order ECP/ATDM applications. A main component of these efforts is the development of ATDM-specific physics enhancements in the finite element algorithms in MFEM and the MFEM-based BLAST Arbitrary Lagrangian-Eulerian (ALE) code [242], in order to provide efficient discretization components for LLNL's ATDM efforts, including the MARBL application (ECP's LLNLApp).

A second main task in the MFEM project is the development of unique unstructured adaptive mesh refinement (AMR) algorithms in MFEM, that focus on generality, parallel scalability, and ease of integration in unstructured mesh applications. The new AMR capabilities can benefit a variety of ECP apps that use unstructured meshes, as well as many other applications in industry and the SciDAC program.

Another aspect of the work is the preparation of the MFEM finite element library and related codes for exascale platforms by using mathematical algorithms and software implementations that exploit increasing on-node concurrency targeting multiple complex architectures (e.g. GPUs). This part of the project is synergistic with and leverages efforts from the ECP CEED co-design center.

MFEM is an open-source finite element library with 3000 downloads/year from 70+ countries. It is freely available at `mfem.org`, on GitHub at `github.com/mfem`, where the MFEM community includes more than 165 members), as well as via Spack and OpenHPC. The application outreach and the integration in the ECP ecosystem is further facilitated by MFEM's participation in ECP's xSDK project.

RAJA, **CHAI**, and **Umpire** are providing software libraries that enable application and library developers to meet advanced architecture portability challenges. The project goals are to enable writing performance portable computational kernels and coordinate complex heterogeneous memory resources among components in a large integrated application. These libraries enhance developer productivity by insulating them from much of the complexity associated with parallel programming model usage and system-specific memory concerns.

The software products provided by this project are three complementary and interoperable libraries:

1. **RAJA**: Software abstractions that enable C++ developers to write performance portable (i.e., single-source) numerical kernels (loops).

2. **CHAI**: C++ "managed array" abstractions that enable transparent and automatic copying of application data to memory spaces at run time as needed based on RAJA execution contexts.

3. **Umpire**: A portable memory resource management library that provides a unified high-level API in C++, C and FORTRAN for resource discovery, memory provisioning, allocation, transformation, and introspection.

Capabilities delivered by these software efforts are needed to manage the diversity and uncertainty associated with current and future HPC architecture design and software support. Moving forward, ECP applications and libraries need to achieve performance portability: without becoming bound to particular (potentially limiting) hardware or software technologies, by insulating numerical algorithms from platform-specific data and execution concerns, and without major disruption as new machine, programming models, and vendor software become available.

These libraries in development in this project are currently used in production ASC applications at Lawrence Livermore National Laboratory (LLNL) and receive most of their support from the LLNL national security application project. They are also being used or being explored/adopted by several ECP application and library projects, including: LLNL ATDM application, GEOS (Subsurface), SW4 (EQSIM), MFEM (CEED co-design), DevilRay (Alpine), and SUNDIALS.

Flux [243, 244] is a next-generation resource management and scheduling software framework under active development at LLNL. This ECP project significantly augments the design and development of this framework to address two specific technical challenges pertaining to exascale computing.

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

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

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

- STAT (highly scalable lightweight debugging tool);
- Archer (low-overhead OpenMP data race detector);
- ReMPI/NINJA (scalable record-and-replay and smart noise injector for MPI); and
- FLiT/FPUChecker (floating-point correctness checking tool suite).

Major efforts include developing and deploying additional capabilities within the team's toolset for exascale systems and integrating them to ASC and ECP/ATDM codes. The team strives to do this through co-design efforts with both large HPC code teams and exascale computing hardware vendors themselves.

Caliper is a program instrumentation and performance measurement framework. It is designed as a performance analysis toolbox in a library, allowing one to bake performance analysis capabilities directly into applications and activate them at runtime. Caliper can be used for lightweight always-on profiling or advanced performance engineering use cases, such as tracing, monitoring, and auto-tuning. It is primarily aimed at HPC applications, but works for any C/C++/Fortran program on Unix/Linux.

Key Challenges

Spack: Spack makes HPC software complexity manageable. Obtaining optimal performance on supercomputers is a difficult task; the space of possible ways to build software is combinatorial in size, and software reuse is hindered by the complexity of integrating a large number of packages and by issues such as binary compatibility. Spack makes it easy to build optimized, reproducible, and reusable HPC software.

MFEM: The key challenges addressed by the LLNL ATDM Mathematical Libraries project are:
Robust high-order finite element methods for ALE compressible flow. While high-order methods offer significant advantages in terms of HPC performance, their application to complicated ALE problems requires careful considerations to control oscillations and ensure accuracy.
Scalable algorithms for unstructured adaptive mesh refinement. Adaptive mesh refinement is a common way to increasing application efficiency in problems with localized features. While block-structured

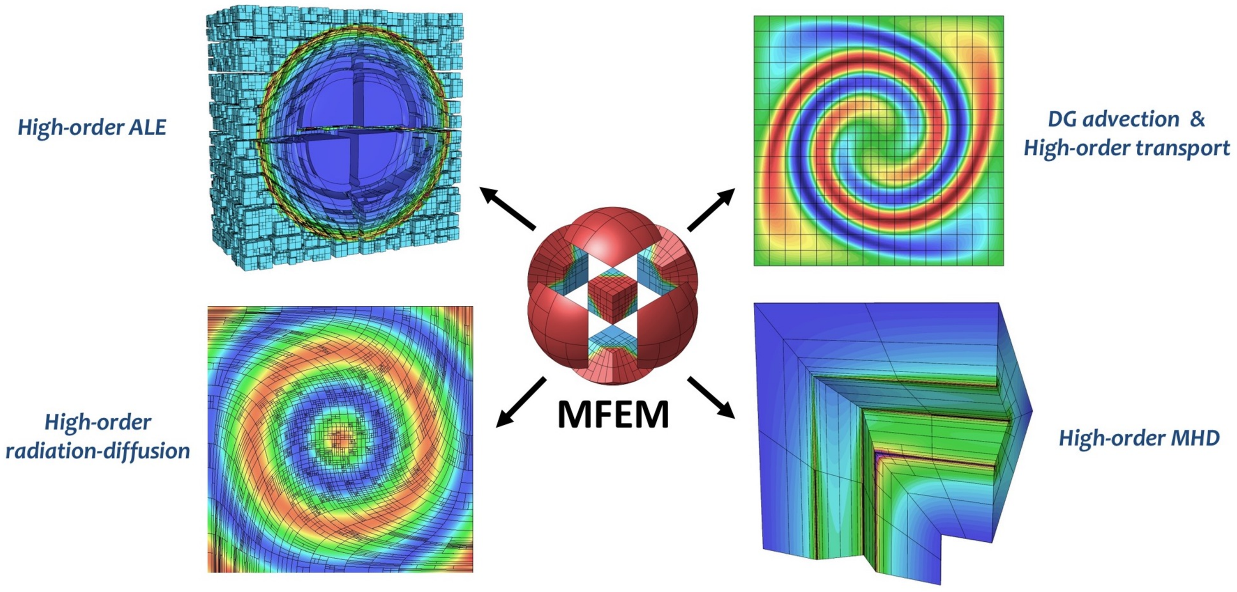

High-order ALE
DG advection & High-order transport
MFEM
High-order radiation-diffusion
High-order MHD

Figure 73: AMR implementation in MFEM allows many applications to benefit from non-conforming adaptivity, without significant changes in their codes.

AMR has been well-studied, applying AMR in unstructured settings is challenging, especially in terms of derefinement, anisotropic refinement, parallel rebalance and scalability.
GPU porting of finite element codes. Due to the relatively high complexity of the finite element machinery, MFEM, BLAST and related codes use object-oriented C++ design that allows generality and flexibility, but poses challenges in terms of porting to GPU architectures. Finding the right balance between generality and performance in the GPU context is an important challenge for many finite element-based codes that remains outstanding in the current software and programming model environment.

RAJA/Umpire/CHAI: Exascale machines are expected to be very diverse, with different GPU, threading, memory models, and node architectures. A parallelization strategy that works well for one machine may not work well for another, but application developers cannot afford to develop multiple versions of their code for each machine they support. Rather, the application must be written using higher-level abstractions, and adapted at a lower level, with minimal programmer effort, to specific machines. RAJA, Umpire, and CHAI address this by giving applications the flexibility to adapt and tune for many target machines, using the same high level kernel formulations. In other words they separate the concerns of performance and correctness and avoid a combinatorial explosion of code versions for the exascale ecosystem.

In addition to performance portability, RAJA, Umpire, and CHAI specifically target the porting issues faced by legacy codes. Where other performance portability frameworks may require a larger up-front investment in data structures and code restructuring, RAJA, Umpire, and CHAI are non-invasive and allow codes to adopt strategies for loop parallelism, data layout tuning, and memory management separately. Legacy applications need not adopt all three at once; they can gradually integrate each framework, at their own pace, with a minimal set of code modifications.

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

AID: Debugging parallel applications running on supercomputers is extremely challenging. greater challenges. Supercomputers may contain very high numbers of compute cores and multiple GPUs, and applications running on such systems must rely on multiple communication and synchronization mechanisms as well as compiler optimization options to effectively utilize the hardware resources. These complexities often produce errors that occur only occasionally, even when run with the exact same input on the same hardware. These so-called non-deterministic bugs are remarkably challenging to catch due in large part to difficulty in reproducing them. Some errors may not even reproduce when being debugged, as the act of debugging may perturb the execution enough to mask the bug. To find and fix these errors, programmers currently must devote a large amount of effort and machine time.

Caliper: Caliper addresses the challenges of providing *meaningful* measurements for large applications. Often, measurements of FLOPs, timings, data movement, and other quantities are not associated with key application constructs that give them meaning. For example, we may know the number of floating point instructions over an entire application run, but if we do not know the number of mesh elements or the particular physics phase associated with the measurement, we may be unable to determine whether the FLOPS achieved are good or bad. Caliper separates these concerns: application developers can instrument the phases other context in their code, and performance analysts and users may turn on performance measurements that are then associated with the context. Caliper associates meaning with HPC performance measurements.

Solution Strategy

Spack: Spack provides a domain-specific language for templated build recipes. It provides a unique infrastructure called the *concretizer*, which solves the complex constraint problems that arise in HPC dependency resolution. Developers can specify builds *abstractly*, and Spack automates the tedious configuration process and drives the build. Spack also includes online services to host recipes, code, and binaries for broad reuse. These repositories are maintained by Spack's very active community of contributors.

MFEM: The MFEM team has performed and documented a lot of research in high-performance mathematical algorithms and finite element discretizations of interest to ATDM applications [245, 246, 247, 248, 249, 250, 251, 252]. Our work has demonstrated that the high-order finite element approach can successfully handle coupled multi-material ALE, radiation-diffusion and MHD. We have also shown how high-order methods can be adapted for monotonicity (positivity preservation), handling of artificial viscosity (shock capturing), sub-zonal physics via closure models, etc.

To enable many applications to take advantage of unstructured mesh adaptivity, the MFEM team is developing AMR algorithms at library level, targeting both *conforming* local refinement on simplex meshes and *non-conforming* refinement for quad/hex meshes. Our approach is fairly general, allowing for any high-order finite element space, H1, H(curl), H(div), on any high-order curved mesh in 2D and 3D, arbitrary order hanging nodes, anisotropic refinement, derifenement and parallel load balancing. An important feature of our library approach is that it is independent of the physics, and thus easy to incorporate in apps, see Figure 73.

As part of the efforts in the ECP co-design Center for Efficient Exascale Discretizations (CEED), the MFEM team is also developing mathematical algorithms and software implementations for finite element methods that exploit increasing on-node concurrency targeting multiple complex architectures (e.g. GPUs). This work includes the libCEED low-level API library, the Laghos miniapp, and several other efforts available through CEED.

To reach its many customers and partners in NNSA, DOE Office of Science, academia and industry, the MFEM team delivers regular releases on GitHub (e.g., mfem-4.0 in 2019 and mfem-4.1 in 2020) that include detailed documentation and many example codes. Code quality is ensured by smoke tests with Travis CI on Linux, Mac, Windows and nightly regression testing at LLNL.

RAJA/Umpire/CHAI: RAJA, Umpire, and CHAI leverage the abstraction mechanisms available in modern C++ (C++11 and higher) compilers, such as Lambdas, policy templates, and constructor/destructor (RAII) patterns for resource management. They aim to provide performance portability at the *library* level, and they do not require special support from compilers. Targeting this level of the software stack gives DOE developers the flexibility to leverage standard parallel programming models like CUDA and OpenMP, without strictly *depending* on robust compiler support for these APIs. If necessary features are unavailable in compilers, library authors are not dependent on vendors for support, and they do not need to wait for these programming models to be fully implemented. These libraries allow applications to work correctly and performantly even if some functionality from OpenMP, CUDA, threading, etc. is missing.

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

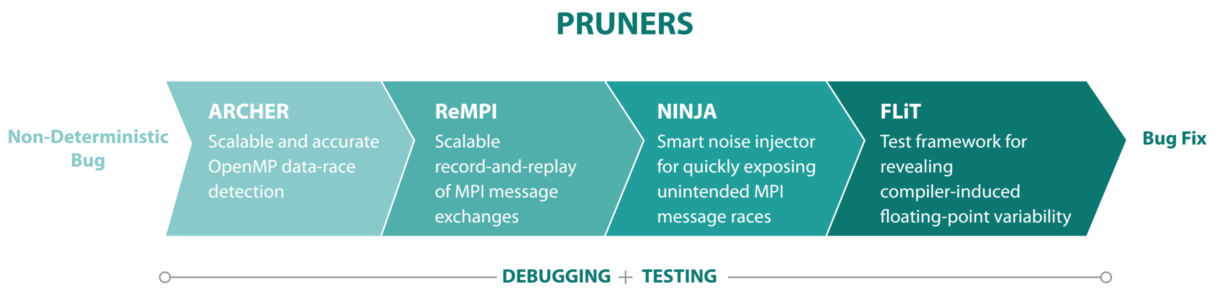

PRUNERS
Non-Deterministic Bug
ARCHER
Scalable and accurate OpenMP data-race detection
ReMPI
Scalable record-and-replay of MPI message exchanges
NINJA
Smart noise injector for quickly exposing unintended MPI message races
FLiT
Test framework for revealing compiler-induced floating-point variability
Bug Fix
DEBUGGING + TESTING

Figure 74: STAT, Archer, NINJA, and FliT: a continuum of debugging tools for exascale.

AID: Debugging a parallel code can be extremely difficult, and the most exhaustive approaches for finding errors can require a large amount of time to run. For example, understanding all of the potential interleavings of parallel threaded code requires combinatorial runtime with respect to the number of threads. It is not feasible to run this type of analysis at all times.

Our strategy is to provide a continuum of debugging tools – from the lightweight tools like STAT, which require only seconds to run and gives a high level overview of a code, to Archer, which requires lightweight code instrumentation, to replay-based fuzzing tools like ReMPI and FLiT, which run the code in a number of configurations to detect errors. With a suite of tools, we can enable developers to find the most common bugs quickly, while still being able to detect deep, hard-to-find issues given sufficient runtime and resources.

Caliper: Caliper is implemented as a C++ library and is linked with applications. Application teams integrate it with their code by adding Caliper annotations at the application level. Contrast this with binary analysis and DWARF line mappings used by most performance tools, which are obtained automatically but increase tool complexity and are typically *not* linked with the application for regular runs.

Applications, their libraries, physics modules, and even runtime systems can be instrumented with Caliper and measured at the same time. All of these layers of the application stack provide additional context to Caliper measurements and enable deeper analysis of the relationships between different parts of the code.

Recent Progress

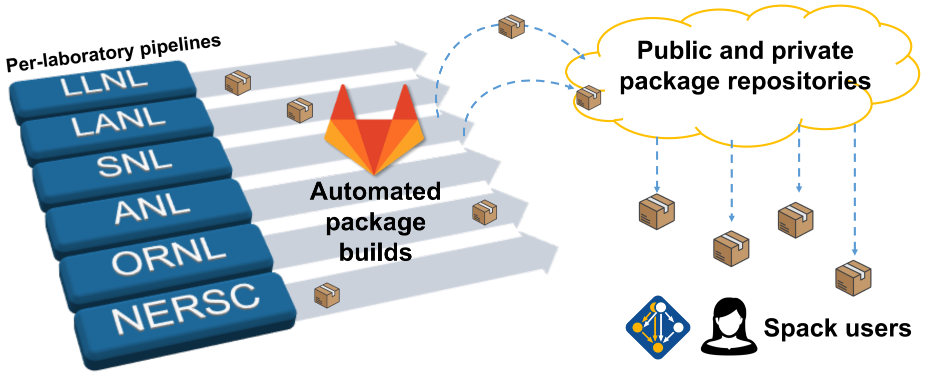


Figure 75: Spack build pipelines at facilities will provide HPC-native binary builds for users.

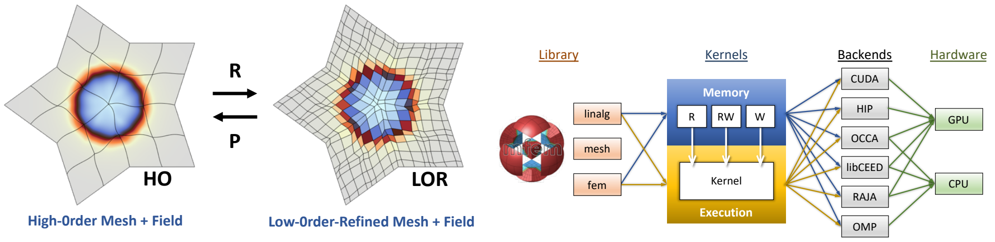


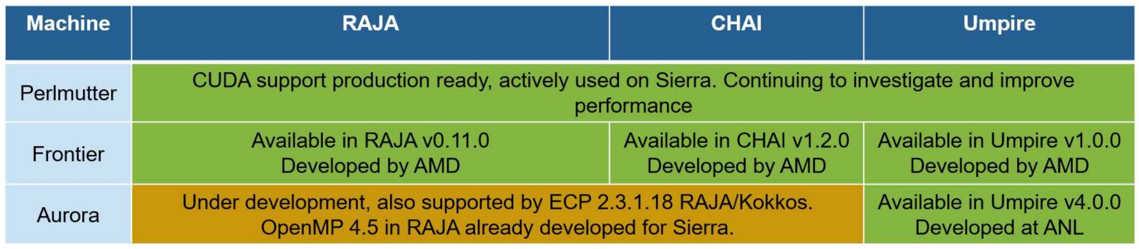
Figure 76: The MFEM team has developed High-Order ↔ Low-Order Transformations and GPU support for many linear algebra and finite element operations

Spack:

- Produced two major software releases: Spack 0.14.0, 0.15.0, as well as multiple bug-fix releases.
- Developed a new Spack containerize command, which can automatically generate multi-stage container build recipes from a Spack environment.
- Completed 100% file system lock-based parallel build capability for Spack. This capability handles both parallel build use case and on-node synchronization.
- Worked with the tri-lab Computing Environment 2 (TCE2) team and code teams to provide Spack support and to add features for these environments. Notable achievements included hardening for Cray platforms, new features in Spack stacks, and GitLab pipeline automation for LC.
- Deployed pull request build testing on GitHub for the first time; the entire E4S stack is now built and tested when GitHub pull requests modify parts of it.

MFEM: Selected recent highlights:

- Completed major software releases: Laghos-3.0, Remhos-1.0, MFEM-4.0, and MFEM-4.1.
- Actively engaged with GPUs and advanced partial assembly algorithms work in MARBL. Demonstrated large-scale GPU performance in ALE hydro in MFEM benchmarks, Laghos, and BLAST on Lassen.
- Implemented ALE discretization improvements for compressible flow in BLAST, including significant improvements in mesh optimization methods, dynamic Adaptive Mesh Refinement (AMR), and transfer between high-order and low-order-refined simulations.



Machine	RAJA	CHAI	Umpire
Perlmutter	CUDA support production ready, actively used on Sierra. Continuing to investigate and improve performance		
Frontier	Available in RAJA v0.11.0 Developed by AMD	Available in CHAI v1.2.0 Developed by AMD	Available in Umpire v1.0.0 Developed by AMD
Aurora	Under development, also supported by ECP 2.3.1.18 RAJA/Kokkos. OpenMP 4.5 in RAJA already developed for Sierra.		Available in Umpire v4.0.0 Developed at ANL

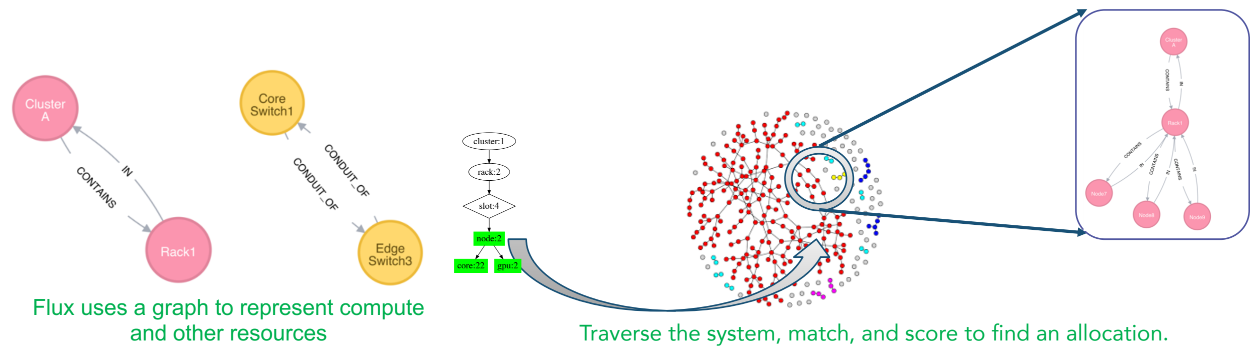
Figure 77: Status of RAJA, Umpire, and CHAI support for exascale platforms.

RAJA/Umpire/CHAI:

- Developed and released support for AMD GPUs (via the HIP programming model) with RAJA, CHAI, and Umpire in preparation for El Capitan.
- Continued to support integration of Umpire into multiple ASC application codes.
- Developed and delivered new capabilities to ASC applications in support of ASC milestones and user needs.
- Developed prototype of fully overlapped asynchronous GPU kernel execution and data transfer to optimize application performance.
- Developed novel memory pool algorithms in Umpire to handle very large allocation counts.

Flux:

- Developed hierarchical scheduling tools within Flux via co-design with UQ Pipeline, demonstrating improvements across the entire lifecycle of simulation campaigns; and demonstrated that DYAD can optimize ML workflows by automatically transferring data files using Flux.
- Demonstrated that Fluxion, our graph-based scheduler, can uniquely enable key use cases of multitiered storage on El Capitan, driving co-design with HPE; tested AMD GPU scheduling by enabling COVID-19


Cluster A
Rack1
IN
CONTAINS
Core Switch1
Edge Switch3
CONDUIT_OF
CONDUIT_OF
Flux uses a graph to represent compute and other resources
cluster:1
rack:2
slot:4
node:2
core:22
gpu:2
Traverse the system, match, and score to find an allocation.

workflows on Corona; demonstrated the viability of power management within Flux using Variorum, a platform-independent power library; and released MPIBIND-2.

AID:

- Demonstrated that our new correctness tools (FLiT and FPChecker) can analyze large LLNL code, discovering previously unknown issues in LLNL code; and facilitated tools co-design via the El Capitan tools working group.

Caliper:

- Integrated the Caliper performance analysis tools into key LLNL applications. Supported existing integration with MARBL via improvements to performance visualizations and analysis capabilities.

Next Steps

Spack: In FY21, the team will focus on:

- Work with MARBL and other code teams to improve developer workflows support for Spack as a development tool.
- Improve integration testing for LLNL code teams by enabling testing for pull requests on platforms of interest to LC.
- Improve Spack support for GPUs, programming models, and exascale architectures.

MFEM:

- Develop adaptive discretizations for high-order finite element ATDM applications, including algorithms such as dynamic AMR, mesh optimization, and hp-adaptivity.
- Support MFEM-based ATDM applications in their transition to exascale hardware.
- Continue engagement with ATDM application work, develop mini-apps, and provide support.

RAJA/Umpire/CHAI:

- Develop and deliver new RAJA capabilities to LLNL codes to support ASC milestone and user requirements, including expanding support for overlapping kernel execution and data transfer and optimized AMD support.

- Continue to develop and deliver Umpire capabilities to applications in support of ASC milestone and user requirements, including interprocess shared memory to support shared on-node data and expanding functionality available inside GPU kernels.
- Continue interactions with El Capitan Center of Excellence (COE) partners to resolve performance and correctness issues identified as application users begin testing on early access hardware.

Flux:

- Demonstrate the scalable end-to-end support within Flux for: 1) smart multitiered storage scheduling and management, 2) CTI-based tool launching, 3) Variorum-enabled power monitoring/capping, 4) more versatile UQ/V&V/ML workflow scheduling, and 5) DYAD-enabled workflow data scheduling on El Capitan early access systems.

AID:

- Implement, evaluate and harden a multilevel general-purpose graphics processing unit (GPGPU) debugging/code-correctness tool suite on early applications running on Collaboration of Oak Ridge, Argonne, and Livermore (CORAL)-2 early access systems.

Caliper:

- Continue SPOT and Caliper application integrations in LLNL applications. Expand tools with more GPU and MPI measurement capabilities.
- Continue support, porting, and deployment of Caliper.

4.6.7 WBS 2.3.6.03 SNL ATDM Software Technologies

Overview

The SNL ATDM Software Technologies projects are now aggregated to include Kokkos, Kokkos kernels, VTK-m, and Operating Systems and On-Node Runtime efforts.

The Kokkos programming model and C++ library enable performance portable on-compute-node parallelism for HPC/exascale C++ applications. Kokkos has been publicly available at http://github.com/kokkos/kokkos since May 2015 and is being used and evaluated by projects at DOE laboratories, PSAAP-II centers, other universities, and organizations such as DoD laboratories. Kokkos library implementation consists of a portable application programmer interface (API) and architecture specific back-ends, including OpenMP, Intel Xeon Phi, and CUDA on NVIDIA GPU. These back-ends are developed and optimized as new application-requested capabilities are added to Kokkos, back-end programming mechanisms evolve, and architectures change.

Kokkos Kernels implements on-node shared memory computational kernels for linear algebra and graph operations, using the Kokkos shared-memory parallel programming model. Kokkos Kernels forms the building blocks of a parallel linear algebra library like Tpetra in Trilinos that uses MPI and threads for parallelism, or it can be used stand-alone in ECP applications. Kokkos Kernels supports several Kokkos backends to support architectures like Intel CPUs, KNLs and NVIDIA GPUs. The algorithms and the implementations of the performance-critical kernels in Kokkos Kernels are chosen carefully to match the features of the architectures. This allows ECP applications to utilize high performance kernels and transfers the burden to Kokkos Kernels developers to maintain them in future architectures. Kokkos Kernels also has support for calling vendor provided libraries where there are optimized kernels available.

VTK-m is a toolkit of scientific visualization algorithms for emerging processor architectures. VTK-m supports the fine-grained concurrency for data analysis and visualization algorithms required to drive extreme scale computing by providing abstract models for data and execution that can be applied to a variety of algorithms across many different processor architectures. The ECP/VTK-m project is building up the VTK-m codebase with the necessary visualization algorithm implementations that run across the varied hardware platforms to be leveraged at the exascale. We will be working with other ECP projects, such as ALPINE, to integrate the new VTK-m code into production software to enable visualization on our HPC systems. For the ASC/ATDM program, the VTK-m project will concentrate on support of ATDM applications and ASC's Advanced Technology Systems (ATS) as well as the ASTRA prototype system at Sandia. General information about VTK-m as well as source code can be found at: http://m.vtk.org.

The OS and On-Node Runtime project focuses on the design, implementation, and evaluation of operating system and runtime system (OS/R) interfaces, mechanisms, and policies supporting the efficient execution of application codes on next-generation platforms. Priorities in this area include the development of lightweight tasking techniques that integrate network communication, interfaces between the runtime and OS for management of critical resources (including multi-level memory, non-volatile memory, and network interfaces), portable interfaces for managing power and energy, and resource isolation strategies at the operating system level that maintain scalability and performance while providing a more full-featured set of system services. The OS/R technologies developed by this project will be evaluated in the context of ATDM application codes running at large-scale on ASC platforms. Through close collaboration with vendors and the broader community, the intention is to drive the technologies developed by this project into vendor-supported system software stacks and gain wide adoption throughout the HPC community.

Key Challenges

Kokkos: The many-core revolution in computing is characterized by: (1) a steady increase in the number of cores within individual computer chips; (2) a corresponding decrease in the amount of memory per core that must be shared by the cores of a chip, and, (3), the diversity of computer chip architectures. This diversity is highly disruptive because each architecture imposes different complex and sometimes conflicting requirements on software to perform well on an architecture. Application software development teams are confronted with the dual challenges of: (1) inventing new parallel algorithms for many-core chips, (2) learning the different programming mechanisms of each architecture, and (2), creating and maintaining separate versions of their software specialized for each architecture. These tasks may involve considerable overhead for organizations in terms of time and cost. Adapting application software to changing HPC requirements is already becoming a large expense for HPC users and can be expected to grow as the diversity of HPC architectures continues to rise. An alternative, however, is creating software that is performance portable across current and future architectures.

Kokkos Kernels: There are several challenges associated with the Kokkos Kernels work. Part of the complexity arises because profiling tools are not yet fully mature for advanced architectures and in this context profiling involves the interplay of several factors which require expert judgment to improve performance. Another challenging aspect is working on milestones that span a variety of projects and code bases. There is a strong dependence on the various application code development teams for our own team's success. In addition, we face a constant tension between the need for production ready tools and components in a realm where the state-of-the-art is still evolving.

VTK-m: The scientific visualization research community has been building scalable HPC algorithms for over 15 years, and today there are multiple production tools that provide excellent scalability [253, 187]. That said, there are technology gaps in data analysis and visualization facing ATDM applications as they move to Exascale. As we approach Exascale, we find that we can rely less on disk storage systems as a holding area for all data between production (by the simulation) and consumption (by the visualization and analysis). To circumvent this limitation, we must integrate our simulation and visualization into the same workflow and provide tools that allows us to run effectively and capture critical information.

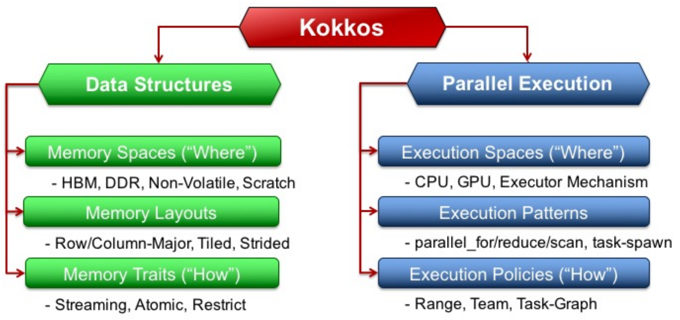
OS & ONR: Exascale challenges for system software span the areas of operating systems, networks, and run time systems. Container technologies are by now ubiquitous in the cloud computing space, but for High Performance Computing their immense potential has been limited by concerns about compatibility with security models and overhead costs. As vendors bring forward new network hardware for exascale, both vendors and application programmers lack insight into how applications actually use networks in practice, especially regarding the characteristics of the messages sent in production codes. As programming models like OpenMP at the node level and MPI at the inter-node level evolve, the particular needs of DOE applications must be addressed in both the development of standards and evaluation of provided run time system implementations.

Solution Strategy

Kokkos: The Kokkos team developed a parallel programming model with flexible enough semantics that it can be mapped on a diverse set of HPC architectures including current multi-core CPUs and massively parallel GPUs. The programming model is implemented using C++ template abstractions, which allow a compile time translation to the underlying programming mechanism on each platform, using their respective primary tool chains. Compared to approaches which rely on source-to-source translators or special compilers, this way leverages the investment of vendors in their preferred programming mechanism without introducing additional, hard to maintain, tools in the compilation chain.

Kokkos Kernels: The Kokkos Kernels team is taking a staged approach to profiling in regards to target architectures and the algorithms involved. We are also coordinating on a regular basis with the other projects that are involved in our work to minimize impediments. In response to the need for production ready tools, we are focusing on a hierarchical approach that involves producing robust, hardened code for core algorithms while simultaneous pursuing research ideas where appropriate.

VTK-m: The VTK-m team is addressing its challenges through development of portable visualization algorithms for VTK-m and leveraging and expanding the Catalyst [187] *in situ* visualization library to apply this technology to ATDM applications on ASC platforms. VTK-m uses the notion of an abstract device adapter, which allows algorithms written once in VTK-m to run well on many computing architectures. The



Kokkos
Data Structures
Parallel Execution
Memory Spaces ("Where")
- HBM, DDR, Non-Volatile, Scratch
Memory Layouts
- Row/Column-Major, Tiled, Strided
Memory Traits ("How")
- Streaming, Atomic, Restrict
Execution Spaces ("Where")
- CPU, GPU, Executor Mechanism
Execution Patterns
- parallel_for/reduce/scan, task-spawn
Execution Policies ("How")
- Range, Team, Task-Graph

Figure 78: Kokkos Execution and Memory Abstractions

device adapter is constructed from a small but versatile set of data parallel primitives, which can be optimized for each platform [171]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [172, 173, 174].

OS & ONR: The OS & ONR team is buying down risk for the use of containers by demonstrating exemplar application containerizations, e.g., for the ATDM SPARC application. We work with facilities staff to develop and implement strategies to deploy containers on our HPC systems and with dev-ops teams to ease the developer burden for code teams seeking to use containers. To better understand network resource utilization, we use an MPI simulator that accepts real network traces of application executions as inputs and provides detailed analysis to inform network hardware vendors and application developers alike. We participate actively in both the OpenMP Language Committee and MPI Forum.

Recent Progress

Kokkos: Kokkos provided a production quality performance portability abstraction to applications and software technology projects under ATDM and ECP which allows them to run on all currently deployed DOE production compute platforms. FY20, the team supported ATDM and ASC IC code Kokkos adoption, debugging and optimization through in house consulting, participation in design discussions, and optimization of Kokkos capabilities. In coordination with Argonne and Oak Ridge National Laboratories, significant progress was made in the development of new SYCL and HIP backends for Kokkos, targeting respective exascale architectures of Intel and AMD. The HIP support is fairly mature, with most commonly used capabilities now available (significant work is continuing to address performance issues). Significant progress was also made on the SYCL backend, but compiler issues have slowed this progress. To better support asynchrony in Kokkos, a prototype for Kokkos-Graphs was developed which uses CUDA graphs under the hoods. It is currently under code review for integration into the mainline Kokkos release. The Kokkos team has also continued its engagement as part of the C++ Standards Committee in support of the DOE HPC community, presented a Linear Algebra proposal to LEWG (the Library Evolution Group) and maturing the MDSPAN implementation.

Kokkos Kernels: The Kokkos Kernels project focused much of its recent work on supporting the L1 milestones for the ATDM SPARC and EMPIRE codes, developing new linear algebra kernels and improving current ones on GPU platforms to support these applications. The primary focus was supporting kernels identified by the Trilinos solver need and continuing the optimization of the Trilinos Tpetra-based solver stack for GPUs. As part of this work, a Kokkos Kernels sparse triangular solver was developed, in support of direct solvers and incomplete factorizations on GPU systems. This implementation performed favorably to vendor implementations (e.g., 2-13x faster than NVIDIA's sparse triangular solver for relevant problems). The team also developed a sparse ILU(k) for better preconditioning options on GPUs and a cluster-coloring based Gauss-Seidel preconditioner that reduces the launch overhead and reduces the number of iterations (this will assist the ATDM EMPIRE application as a smoother). In addition, the Kokkos Kernels project also supported the ATDM GEMMA team by continuing the co-development of GPU based dense solver ADELUS for distributed memory systems, especially Sierra runs. This ADELUS code achieved 7.7 PetaFLOPs of performance when run on 7600 GPUs of Sierra, significantly outperforming existing state of the art solvers.

VTK-m: The VTK-m team made significant improvements to ParaView/Catalyst in support of ATDM applications. These include support for static builds of ParaView/Catalyst, reductions to the memory footprint of Catalyst when used with SPARC, and customized scripting to simplify *in situ* visualization configuration. The team also began development on a performance-evaluation suite that currently consists of a "driver" program that allows the team to test VTK-m and Catalyst performance. The "driver" program can read CGNS and Exodus files and drive Catalyst as if a simulation were producing the data. The code was used to evaluate Catalyst with VTK-m enabled filters on the Vortex GPU hardware. Finally, the team developed and added a Kokkos device adapter to VTK-m. This advance will simplify porting VTK-m to ECP architectures and reduce VTK-m developer time.

OS & ONR: The OS & ONR team had a number of recent accomplishments. The LogGOPSim simulation framework that was enhanced to quantify MPI resource usage was used to characterize ATDM workloads and examine the relationship between MPI resource usage and application performance and scalability. A detailed report examining the communication behavior of the Sandia Parallel Aerodynamics and Reentry Code (SPARC) based primarily on its use of MPI resources was completed. The analysis shows that SPARC's communication behavior compares favorably to two well-studied workloads that have been shown to run efficiently at the scale of our leadership-class systems. We also collaborated with ETH Zurich to merge several recent changes that improve the performance and scalability of the simulator framework into a new open source release. We also investigated the use of unprivileged container builds using Podman to enable building of new container images from the login or service/compute nodes of the target HPC resource directly. Specifically, the team installed and validated the use of Podman in an unprivileged setting on the Sandia Stria system. Initial results showed that the full Sandia ATSE software stack and applications can be built using Podman on Stria, but further improvements and testing will be necessary to move into a full production container build capability. A full report of the updated container workflow model is highlighted in an SC'20 paper entitled "Chronicles of Astra: Challenges and Lessons from the First Petascale Arm Supercomputer." MPI Forum work has progressed in several key improvements for MPI 4.0. Partitioned communication has been officially added and work continues on improvements for GPU communication. We contributed to multiple improvements on collective operations, feedback on dynamic sessions in MPI, and to improvements for 64-bit support for very large messages. We also contributed to reviewing and providing extensive feedback on updates to the underlying semantics of the MPI specification and have contributed some text as well. We continue to actively participate in multiple working groups and co-lead work on partitioned communication and advanced collectives and persistent operations. We are also actively engaged as a key partner with the hybrid programming models group, contributing work on efficient native MPI API support for GPU architectures. The OpenMP Language Committee work focused on preparing for the release of version 5.1 of the specification. Our contributions include leading the task parallelism subcommittee and editing/revising two chapters of the specification to ensure correctness.

Next Steps

Kokkos: The Kokkos project will continue to provide high quality (production) Kokkos support and consultation for ASC applications and libraries. The work on continuing to mature the Kokkos backends will also continue. In particular, the team will be working (in collaboration with Oak Ridge National Laboratory) to mature and optimize the HIP backend for exascale platforms using AMD GPUs, (in collaboration with Argonne National Laboratory) to develop and optimize SYCL/DPC++ backend for the exascale platforms using Intel GPUs, and to mature and Optimize the OpenMP Target backend as an alternative to the primary tool chains on the exascale platforms. A particular driver of this backend development this year will be to demonstrate a working Kokkos-based Trilinos solver stack. The Kokkos team will also continue its engagement with the C++ Standards committee, developing C++17 based API improvements, which will allow the Kokkos Programming Model to be more consistent with C++ and thus reduce the mental load for users. An example of this is using RangePolicy with team handles as nested loop constructs instead of TeamThreadRange. Furthermore, the team will engage the C++ standards committee to further the adoption of successful Kokkos concepts into the standard, and provide feedback on proposed concurrency mechanisms such as the executors proposal. The team will also continue development of the proposed linear algebra capabilities with actual parallel backends, in order to allow an adoption of linear algebra into the standard by 2023.

Kokkos Kernels: The Kokkos Kernels project will continue to develop key optimized kernels for the GPU-based exascale systems and provide high quality support/consultation for ATDM/ASC applications and libraries. The team will develop and deliver a portable MIS-2 algorithm to support better coarsening schemes in multigrid methods on GPU systems. The team plans to explore new algorithms for sparse matrix-matrix multiplication that have emerged in the field and develop portable implementations of this algorithms, especially targeting SIERRA and El Capitan platforms (progress on the El Capitan implementation will depend on the maturity of the HIP toolchain). The team will start developing HIP backend support (targeting El Capitan) for key sparse and dense linear algebra kernels. The team will continue to actively engage the vendor community (NVIDIA, ARM, AMD) to develop and deliver kernels using Kokkos Kernels as reference implementation in order to better support ASC/ATDM codes and the broader CSE community.

VTK-m: The ATDM/VTK-m project has transitioned away from building functionality into the VTK-m toolkit to addressing the needs of other ST projects and ATDM applications. The FY21 work will continue to focus on the three key goals of ECP: performance, integration, and quality. In support of these goals, the team will continue development of a performance-evaluation suite capable of evaluating VTK-m and Catalyst on ATS-2 (or similar) machines; make improvements to the Catalyst Python IDE, enabling the management of ParaView scripts inside Python editing tools such as Jupyter; and continue work to unify and evolve the Phactori scripting language for *in situ* visualization, simplifying configuration of Catalyst for HPC applications.

OS & ONR: We plan to characterize OS noise behavior in the context of containers during execution of ASC workloads and its impact on performance and scalability. There is also interest in understanding the variability of GPU kernel launch latency. We will continue to work with vendors, facilities, and application/library developers to further leverage container technologies for ASC workloads. We will also continue to contribute to the OpenMP and MPI standards bodies to shape the direction of the OpenMP and MPI parallel programming models to provided needed capabilities for ASC workloads.

5. CONCLUSION

ECP ST is providing a collection of essential software capabilities necessary for successful results from Exascale computing platforms, while also delivery a suite of products that can be sustained into the future. This Capabilities Assessment Report and subsequent versions will provide a periodic summary of capabilities, plans, and challenges as the Exascale Computing Project proceeds.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (ECP), Project Number: 17-SC-20-SC, a collaborative effort of two DOE organizations—the Office of Science and the National Nuclear Security Administration—responsible for the planning and preparation of a capable Exascale ecosystem—including software, applications, hardware, advanced system engineering, and early testbed platforms—to support the nation's Exascale computing imperative.

REFERENCES

[1] Todd Gamblin, Matthew P. LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and W. Scott Futral. The Spack Package Manager: Bringing order to HPC software chaos. In *Supercomputing 2015 (SC'15)*, Austin, Texas, November 15-20 2015. LLNL-CONF-669890.

[2] Todd Gamblin, Gregory Becker, Peter Scheibel, Matt Legendre, and Mario Melara. Managing HPC Software Complexity with Spack. In *Exascale Computing Project 2nd Annual Meeting*, Knoxville, TN, February 6-8 2018. Half day.

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

[4] Michael A. Heroux. Episode 17: Making the development of scientific applications effective and efficient. https://soundcloud.com/exascale-computing-project/episode-17-making-the-development-of-scientific-applications-effective-and-efficient.

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

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

[7] Paul Basco. DOE Order 413.3B: Program and Project Management (PM) for the Acquisition of Capital Assets, Significant Changes to the Order. https://www.energy.gov/sites/prod/files/maprod/documents/15-1025_Bosco.pdf.

[8] E4S Community Policies. https://e4s-project.github.io/policies.html.

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

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

[11] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. Fine-grained multithreading support for hybrid threaded MPI programming. *Int. J. High Perform. Comput. Appl.*, 24(1):49–57, February 2010.

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

[13] William Gropp and Ewing Lusk. Fault tolerance in message passing interface programs. *The International Journal of High Performance Computing Applications*, 18(3):363–372, 2004.

[14] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur, and J. L. Traeff. MPI on Millions of Cores. *Parallel Processing Letters (PPL)*, 21(1):45–60, March 2011.

[15] D. Buntinas, B. Goglin, D. Goodell, G. Mercier, and S. Moreaud. Cache-efficient, intranode, large-message MPI communication with MPICH2-nemesis. In *2009 International Conference on Parallel Processing*, pages 462–469, September 2009.

[16] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur and J. L. Traeff. MPI on millions of cores. *Parallel Processing Letters*, 21(1):45–60, 2011.

[17] Y. Guo, C. J. Archer, M. Blocksome, S. Parker, W. Bland, K. Raffenetti, and P. Balaji. Memory compression techniques for network address management in MPI. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1008–1017, May 2017.

[18] Nikela Papadopoulou, Lena Oden, and Pavan Balaji. A performance study of UCX over infiniband. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '17, pages 345–354, Piscataway, NJ, USA, 2017. IEEE Press.

[19] Ken Raffenetti, Abdelhalim Amer, Lena Oden, Charles Archer, Wesley Bland, Hajime Fujita, Yanfei Guo, Tomislav Janjusic, Dmitry Durnov, Michael Blocksome, Min Si, Sangmin Seo, Akhil Langer, Gengbin Zheng, Masamichi Takagi, Paul Coffman, Jithin Jose, Sayantan Sur, Alexander Sannikov, Sergey Oblomov, Michael Chuvelev, Masayuki Hatanaka, Xin Zhao, Paul Fischer, Thilina Rathnayake, Matt Otten, Misun Min, and Pavan Balaji. Why is MPI so slow?: Analyzing the fundamental limits in implementing MPI-3.1. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 62:1–62:12, New York, NY, USA, 2017. ACM.

[20] Ashwin M. Aji, Lokendra S. Panwar, Feng Ji, Karthik Murthy, Milind Chabbi, Pavan Balaji, Keith R. Bisset, James Dinan, Wu chun Feng, John Mellor-Crummey, Xiaosong Ma, and Rajeev Thakur. MPI-ACC: Accelerator-aware MPI for scientific applications. *IEEE Trans. Parallel Distrib. Syst.*, 27(5):1401–1414, May 2016.

[21] Humayun Arafat, James Dinan, Sriram Krishnamoorthy, Pavan Balaji, and P. Sadayappan. Work stealing for GPU-accelerated parallel programs in a global address space framework. *Concurr. Comput. : Pract. Exper.*, 28(13):3637–3654, September 2016.

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

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

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

[25] Mohammad Javad Rashti, Jonathan Green, Pavan Balaji, Ahmad Afsahi, and William Gropp. Multi-core and network aware MPI topology functions. In Yiannis Cotronis, Anthony Danalis, Dimitrios S. Nikolopoulos, and Jack Dongarra, editors, *Recent Advances in the Message Passing Interface*, pages 50–60, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[26] Torsten Hoefler, Rolf Rabenseifner, Hubert Ritzdorf, Bronis R. de Supinski, Rajeev Thakur, and Jesper Larsson Traff. The scalable process topology interface of MPI 2.2. *Concurr. Comput. : Pract. Exper.*, 23(4):293–310, March 2011.

[27] Leonardo Arturo Bautista Gomez Robert Latham and Pavan Balaji. Portable topology-aware MPI-I/O. In *IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2017.

[28] Min Si and Pavan Balaji. Process-based asynchronous progress model for MPI point-to-point communication. In *IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2017.

[29] Pavan Balaji Seyed Hessamedin Mirsadeghi, Jesper Larsson Traff and Ahmad Afsahi. Exploiting common neighborhoods to optimize MPI neighborhood collectives. In *IEEE International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2017.

[30] Camille Coti, Thomas Herault, Pierre Lemarinier, Laurence Pilard, Ala Rezmerita, Eric Rodriguez, and Franck Cappello. Blocking vs. non-blocking coordinated checkpointing for large-scale fault tolerant MPI. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[31] H. V. Dang, S. Seo, A. Amer, and P. Balaji. Advanced thread synchronization for multithreaded MPI implementations. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 314–324, May 2017.

[32] Abdelhalim Amer, Charles Archer, Michael Blocksome, Chongxiao Cao, Michael Chuvelev, Hajime Fujita, Maria Garzaran, Yanfei Guo, Jeff R. Hammond, Shintaro Iwasaki, Kenneth J. Raffenetti, Mikhail Shiryaev, Min Si, Kenjiro Taura, Sagar Thapaliya, and Pavan Balaji. Software Combining

to Mitigate Multithreaded MPI Contention. In *Proceedings of the ACM International Conference on Supercomputing*, ICS '19, pages 367–379, New York, NY, USA, 2019. ACM.

[33] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. Carns, A. Castello, D. Genet, T. Herault, S. Iwasaki, P. Jindal, L. V. Kale, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, and P. Beckman. Argobots: A lightweight low-level threading and tasking framework. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):512–526, March 2018.

[34] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, Sangmin Seo, and Pavan Balaji. BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads. In *Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques*, PACT '19, New York, NY, USA, 2019. ACM.

[35] Rohit Zambre, Aparna Chandramowliswharan, and Pavan Balaji. How I learned to stop worrying about user-visible endpoints and love MPI. In *Proceedings of the 34th ACM International Conference on Supercomputing*, ICS '20, New York, NY, USA, 2020. ACM.

[36] Q. Cao, Y. Pei, T. Herault, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. E. Keyes, and J. Dongarra. Performance Analysis of Tile Low-Rank Cholesky Factorization Using PaRSEC Instrumentation Tools. In *ProTools'19*, ProTools'19, 2019.

[37] Y. Pei, G. Bosilca, I. Yamazaki, A. Ida, and J. Dongarra. Evaluation of Programming Models to Address Load Imbalance on Distributed Multi-Core CPUs: A Case Study with Block Low-Rank Factorization. In *Parallel Applications Workshop, Alternatives To MPI+X*, PAW-ATM 2019, Nov 2019.

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

[39] Thomas Herault, Yves Robert, George Bosilca, Robert J Harrison, Cannada A Lewis, Edward F Valeev, and Jack J Dongarra. Distributed-memory multi-GPU block-sparse tensor contraction for electronic structure (revised version). Research Report RR-9365, Inria - Research Centre Grenoble – Rhône-Alpes, October 2020.

[40] Ole Schütt, Peter Messmer, Jürg Hutter, and Joost VandeVondele. *GPU-Accelerated Sparse Matrix–Matrix Multiplication for Linear Scaling Density Functional Theory*, chapter 8, pages 173–190. John Wiley & Sons, Ltd, 2016.

[41] Chong Peng, Justus A Calvin, Fabijan Pavošević, Jinmei Zhang, and Edward F Valeev. Massively Parallel Implementation of Explicitly Correlated Coupled-Cluster Singles and Doubles Using TiledArray Framework. *J. Phys. Chem. A*, 120(51):10231–10244, December 2016.

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

[43] John Bachan, Scott B. Baden, Steven Hofmeyr, Mathias Jacquelin, Amir Kamil, Dan Bonachea, Paul H. Hargrove, and Hadia Ahmed. UPC++: A High-Performance Communication Framework for Asynchronous Computation. In *Proceedings of the 33rd IEEE International Parallel & Distributed Processing Symposium*, IPDPS. IEEE, 2019. https://doi.org/10.25344/S4V88H.

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

[45] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. Legion: expressing locality and independence with logical regions. In *Proceedings of the international conference on high performance computing, networking, storage and analysis*, page 66. IEEE Computer Society Press, 2012.

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

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

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

[49] Dan Bonachea and Paul H. Hargrove. GASNet-EX: A High-Performance, Portable Communication Library for Exascale. In *Proceedings of Languages and Compilers for Parallel Computing (LCPC'18)*, volume 11882 of *Lecture Notes in Computer Science*. Springer International Publishing, October 2018. Lawrence Berkeley National Laboratory Technical Report (LBNL-2001174). https://doi.org/10.25344/S4QP4W.

[50] Dan Bonachea and Paul Hargrove. GASNet specification, v1.8.1. Technical Report LBNL-2001064, Lawrence Berkeley National Laboratory, August 2017. https://doi.org/10.2172/1398512.

[51] John Bachan, Dan Bonachea, and Amir Kamil. UPC++ v1.0 Specification, Revision 2020.3.0. Technical Report LBNL-2001268, Lawrence Berkeley National Laboratory, March 2020. https://doi.org/10.25344/S4T01S.

[52] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick. UPC++: A PGAS extension for C++. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1105–1114, May 2014. https://doi.org/10.1109/IPDPS.2014.115.

[53] David E. Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E. Grant, Thomas Naughton, Howard P. Pritchard, Martin Schulz, and Geoffroy R. Vallee. A survey of MPI usage in the U.S. Exascale Computing Program. Technical Report ORNL/SPR-2018/790, Oak Ridge National Laboratory, 2018. https://doi.org/10.2172/1462877.

[54] Swann Perarnau, Judicael A Zounmevo, Matthieu Dreher, Brian C Van Essen, Roberto Gioiosa, Kamil Iskra, Maya B Gokhale, Kazutomo Yoshii, and Pete Beckman. Argo NodeOS: Toward unified resource management for exascale. In *Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International*, pages 153–162. IEEE, 2017.

[55] J. S. Vetter, R. Brightwell, M. Gokhale, P. McCormick, R. Ross, J. Shalf, K. Antypas, D. Donofrio, T. Humble, C. Schuman, B. Van Essen, S. Yoo, A. Aiken, D. Bernholdt, S. Byna, K. Cameron, F. Cappello, B. Chapman, A. Chien, M. Hall, R. Hartman-Baker, Z. Lan, M. Lang, J. Leidel, S. Li, R. Lucas, J. Mellor-Crummey, P. Peltz Jr., T. Peterka, M. Strout, and J. Wilke. Extreme heterogeneity 2018 - productive computational science in the era of extreme heterogeneity: Report for DOE ASCR workshop on extreme heterogeneity. Technical report, USDOE Office of Science (SC) (United States), 2018.

[56] Daniel Barry, Anthony Danalis, and Heike Jagode. Effortless monitoring of arithmetic intensity with papi's counter analysis toolkit. In *13th International Workshop on Parallel Tools for High Performance Computing*, Dresden, Germany, 2020-09 2020. Springer International Publishing, Springer International Publishing.

[57] Heike Jagode, Anthony Danalis, and Jack Dongarra. Formulation of requirements for new papi++ software package: Part i: Survey results. Technical Report 1, ICL-UT-20-02, 2020-01 2020.

[58] Heike Jagode, Anthony Danalis, and Damien Genet. Roadmap for refactoring classic papi to papi++: Part ii: Formulation of roadmap based on survey results. Technical Report 2, ICL-UT-20-09, 2020-07 2020.

[59] J.E. Denny, S. Lee, and J.S. Vetter. Clacc: Translating OpenACC to OpenMP in Clang. In *2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, Dallas, TX, USA, 2018. IEEE.

[60] Jungwon Kim, Kittisak Sajjapongse, Seyong Lee, and Jeffrey S. Vetter. Design and Implementation of Papyrus: Parallel Aggregate Persistent Storage. In *Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium*, IPDPS '17, pages 1151–1162, 2017.

[61] Jungwon Kim, Seyong Lee, and Jeffrey S. Vetter. PapyrusKV: A high-performance parallel key-value store for distributed NVM architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 57:1–57:14, 2017.

[62] Johannes Doerfert, Joseph Huber, Stefan Stipanovic, Giorgis Georgakoudis, Hamilton Tobon Mosquera, and Shilei Tian. (OpenMP) Parallelism Aware Optimizations. https://whova.com/embedded/session/llvm_202010/1162344/, 2020.

[63] Giorgis Georgakoudis, Johannes Doerfert, Ignacio Laguna, and Thomas R. W. Scogland. FAROS: A Framework to Analyze OpenMP Compilation Through Benchmarking and Compiler Optimization Analysis. In Kent Milfeld, Bronis R. de Supinski, Lars Koesterke, and Jannis Klinkenberg, editors, *OpenMP: Portable Multi-Level Parallelism on Modern Systems*, pages 3–17, Cham, 2020. Springer International Publishing.

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

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

[66] Michael Kruse, Hal Finkel, and Xingfu Wu. Autotuning Search Space for Loop Transformations. *LLVM-HPC Workshop @SC*, 2020.

[67] Hal Finkel, Alex McCaskey, Tobi Popoola, Dmitry Lyakh, and Johannes Doerfert. *Really* Embedding Domain-Specific Languages into C++. *LLVM-HPC Workshop @SC*, 2020.

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

[69] Kyle Friedline, Sunita Chandrasekaran, M. Graham Lopez, and Oscar Hernandez. OpenACC 2.5 Validation Testsuite Targeting Multiple Architectures. In *High Performance Computing*, pages 557–575, Cham, 2017. Springer International Publishing.

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

[71] Tuowen Zhao, Samuel Williams, Mary Hall, and Hans Johansen. Delivering performance portable stencil computations on cpus and gpus using bricks. In *Proceedings of the International Workshop on Performance, Portability and Productivity in HPC (P3HPC), SC'18*, Nov 2018.

[72] T. Zhao, P. Basu, S. Williams, M. Hall, and H. Johansen. Exploiting reuse and vectorization in blocked stencil computations on CPUs and GPUs. In *accepted and to appear, SC 2019*, Nov 2019.

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

[74] GWT-TUD GmbH. Vampir. https://vampir.eu, 2020. Accessed: 2020-10-15.

[75] Jungwon Kim Kim and Jeffrey S. Vetter. Implementing Efficient Data Compression and Encryption in a Persistent Key-value Store for HPC. *The International Journal of High Performance Computing Applications*, 33(6):1098–1112, 2019.

[76] Bronis R. De Supinski and Michael Klemm. OpenMP Technical Report 8: Version 5.0 Preview 2. http://www.openmp.org/wp-content/uploads/openmp-tr8.pdf, 2017. [Online; accessed 31-November-2019].

[77] Oleksandr Zinenko, Sven Verdoolaege, Chandan Reddy, Jun Shirako, Tobias Grosser, Vivek Sarkar, and Albert Cohen. Modeling the conflicting demands of parallelism and temporal/spatial locality in affine scheduling. In *Proceedings of the 27th International Conference on Compiler Construction*, pages 3–13. ACM, 2018.

[78] Jose Manuel Monsalve Diaz, Kyle Friedline, Swaroop Pophale, Oscar Hernandez, David Bernholdt, and Sunita Chandrasekaran. Analysis of OpenMP 4.5 Offloading in Implementations: Correctness and Overhead. *Parallel Computing*, page 102546, 08 2019.

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

[80] Alok Mishra, Martin Kong, and Barbara Chapman. Kernel Fusion/Decomposition for Automatic GPU-offloading. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, CGO 2019, pages 283–284, Piscataway, NJ, USA, 2019. IEEE Press.

[81] Lingda Li and Barbara Chapman. Compiler Assisted Hybrid Implicit and Explicit GPU Memory Management under Unified Address Space. In *Proceedings of the 31st ACM/IEE International Conference for High Performance Computing, Networking, Storage, and Analysis in Denver, CO, USA (SC '19)*, November 2019.

[82] Michael Kruse and Hal Finkel. Design and Use of Loop Transformation Pragmas. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, 2019.

[83] Vinu Sreenivasan, Rajath Javali, Mary W. Hall, Prasanna Balaprakash, Thomas R. W. Scogland, and Bronis R. de Supinski. A Framework for Enabling OpenMP Autotuning. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 50–60, 2019.

[84] V Sreenivasan, R Javali, M Hall, P Balaprakash, and B R de Supinski. A Framework for Enabling OpenMP Autotuning. 11718, 6 2019.

[85] Thomas R. W. Scogland, Dan Sunderland, Stephen L. Olivier, David S. Hollman, Noah Evans, and Bronis R. de Supinski. Making OpenMP Ready for C++ Executors. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 320–332, 2019.

[86] Yonghong Yan, Anjia Wang, Chunhua Liao, Thomas R. W. Scogland, and Bronis R. de Supinski. Extending OpenMP Metadirective Semantics for Runtime Adaptation. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 201–214, 2019.

[87] Vivek Kale, Christian Iwainsky, Michael Klemm, Jonas H. Müller Korndörfer, and Florina M. Ciorba. Toward a Standard Interface for User-Defined Scheduling in OpenMP. In *OpenMP: Conquering the Full Hardware Spectrum - 15th International Workshop on OpenMP, IWOMP 2019, Auckland, New Zealand, September 11-13, 2019, Proceedings*, pages 186–200, 2019.

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

[89] Jonas H. Muller Kondorfer, Florina Ciorba, Christian Iwainsky, Johannes Doerfert, Hal Finkel, Vivek Kale, and Michael Klemm. A Runtime Approach for Dynamic Load Balancing of OpenMP Parallel Loops in LLVM. In *Proceedings of the 31st ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis in Denver, CO, USA (SC '19)*, November 2019.

[90] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, Sangmin Seo, and Pavan Balaji. BOLT: Optimizing OpenMP Parallel Regions with User-Level Threads. In *Proceedings of the 28th International Conference on Parallel Architectures and Compilation Techniques (PACT '19)*, September 2019.

[91] Alok Mishra, Abid M. Malik, and Barbara Chapman. Using Machine Learning for OpenMP GPU Offloading in LLVM. In *ACM SRC to be held at SC20.*

[92] Alok Mishra, Lingda Li, Martin Kong, Hal Finkel, and Barbara Chapman. Benchmarking and Evaluating Unified Memory for OpenMP GPU Offloading. In *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC, LLVM-HPC@SC 2017, Denver, CO, USA, November 13, 2017*, pages 6:1–6:10, 2017.

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

[94] K. Taura and Akinori Yonezawa. Fine-grain multithreading with minimal compiler support – a cost effective approach to implementing efficient multithreading languages. In *PLDI*, pages 320–333, 1997.

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

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

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

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

[99] A. Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *SenSys*, pages 29–42, 2006.

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

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

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

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

[104] R. von Behren, Jeremy Condit, Feng Zhou, George C. Necula, and Eric Brewer. Capriccio: Scalable threads for internet services. In *SOSP*, pages 268–281, 2003.

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

[106] P. Li and Steve Zdancewic. Combining events and threads for scalable network services implementation and evaluation of monadic, application-level concurrency primitives. In *PLDI*, pages 189–199, 2007.

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

[108] J. del Cuvillo, Weirong Zhu, Ziang Hu, and Guang R. Gao. TiNy threads: A thread virtual machine for the Cyclops64 cellular architecture. In *WMPP*, 2005.

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

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

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

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

[113] Sangmin Seo, Abdelhalim Amer, Pavan Balaji, Cyril Bordage, George Bosilca, Alex Brooks, Philip Carns, Adrián Castelló, Damien Genet, Thomas Herault, et al. Argobots: a lightweight low-level threading and tasking framework. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):512–526, 2018.

[114] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, and Pavan Balaji. Lessons learned from analyzing dynamic promotion for user-level threading. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, page 23. IEEE Press, 2018.

[115] Shintaro Iwasaki, Abdelhalim Amer, Kenjiro Taura, and Pavan Balaji. Analyzing the performance trade-off in implementing user-level threads. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1859–1877, 2020.

[116] Abdelhalim Amer Amer, Milind Chabbi, Huiwei Lu, Yanji Wei, Jeff Hammond, Satoshi Matsuoka, and Pavan Balaji. Lock contention management in multithreaded mpi. *ACM Transactions on Parallel Computing*, 2018.

[117] H. V. Dang, S. Seo, A. Amer, and P. Balaji. Advanced thread synchronization for multithreaded mpi implementations. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 314–324, May 2017.

[118] Abdelhalim Amer, Huiwei Lu, Yanjie Wei, Pavan Balaji, and Satoshi Matsuoka. Mpi+threads: Runtime contention and remedies. *SIGPLAN Not.*, 50(8):239–248, January 2015.

[119] N. Evans, J. Ciesko, S. Olivier, H. Pritchard, S. Iwasaki, K. Raffenetti, and P. Balaji. Implementing flexible threading support in Open MPI. In *Proceedings of the Workshop on Exascale MPI in conjunction with International Conference for High Performance Computing, Networking, Storage, and Analysis (ExaMPI '20)*, Nov. 2020.

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

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

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

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

[124] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, J. Loe, P. Luszczek, S. Pranesh, S. Rajamanickam, T. Ribizel, B. F. Smith, K. Swirydowicz, S. Thomas, S. Tomov, Y. M. Tsai, and U. M. Yang. A Survey of Numerical Linear Algebra Methods Utilizing Mixed Precision Arithmetic, 2020. Submitted to International Journal on High Performance Computing.

[125] Hartwig Anzt, Erik Boman, Rob Falgout, Pieter Ghysels, Michael Heroux, Xiaoye Li, Lois Curfman McInnes, Richard Tran Mills, Sivasankaran Rajamanickam, Karl Rupp, Barry Smith, Ichitaro Yamazaki, and Ulrike Meier Yang. Preparing Sparse Solvers for Exascale Computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378, January 2020.

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

[127] xSDK Community Package Policies. https://github.com/xsdk-project/xsdk-community-policies.

[128] xSDK Examples. https://github.com/xsdk-project/xsdk-examples.

[129] GPTune. https://github.com/gptune/GPTune.

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

[131] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, D. Karpeyev, D. Kaushik, M. Knepley, D. May, L. Curfman McInnes, R. Mills, T. Munson, K. Rupp, P. Sanan, B. Smith, S. Zampini, H. Zhang, and H. Zhang. PETSc Users Manual Revision 3.14. Technical Memorandum ANL-95/11 Rev. 3.14, Argonne National Laboratory, 2020.

[132] SUNDIALS Project Team. SUNDIALS Web page. http://computation.llnl.gov/projects/sundials.

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

[134] R. D. Falgout, J. E. Jones, and U. M. Yang. The design and implementation of *hypre*, a library of parallel high performance preconditioners. In A. M. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering*, chapter 8, pages 267–294. Springer-Verlag, 2006. UCRL-JRNL-205459.

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

[136] S. F. Ashby and R. D. Falgout. A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. *Nuclear Science and Engineering*, 124(1):145–159, September 1996. UCRL-JC-122359.

[137] R. Li, B. Sjogreen, and U. M. Yang. A new class of amg interpolation operators based on matrix-matrix operations. 2020. Submitted to SIAM Journal on Scientific Computing.

[138] Stanimire Tomov, Azzam Haidar, Alan Ayala, Hejer Shaiek, and Jack Dongarra. Fft-ecp implementation optimizations and features phase. ECP WBS 2.3.3.09 Milestone Report ICL-UT-19-12, FFT-ECP ST-MS-10-1440, 2019-10 2019.

[139] Alan Ayala, Stanimire Tomov, Xi Luo, Hejer Shaiek, Azzam Haidar, George Bosilca, and Jack Dongarra. Impacts of multi-gpu mpi collective communications on large fft computation. In *SC'19, Proc. of Workshop on Exascale MPI (ExaMPI)*, Denver, CO, 2019.

[140] Hejer Shaiek, Stanimire Tomov, Alan Ayala, Azzam Haidar, and Jack Dongarra. Gpudirect mpi communications and optimizations to accelerate ffts on exascale systems. Extended Abstract icl-ut-19-06, 2019-09 2019.

[141] Azzam Haidar, Stanimire Tomov, Jack Dongarra, and Nicholas J. Higham. Harnessing gpu tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, pages 47:1–47:11, Piscataway, NJ, USA, 2018. IEEE Press.

[142] A. Sorna, X. Cheng, E. D'Azevedo, K. Won, and S. Tomov. Optimizing the fast fourier transform using mixed precision on tensor core hardware. In *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, pages 3–7, Dec 2018.

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

[144] Rajib Nath, Stanimire Tomov, and Jack Dongarra. An improved magma gemm for fermi graphics processing units. *Int. J. High Perform. Comput. Appl.*, 24(4):511–515, November 2010.

[145] Jakub Kurzak, Stanimire Tomov, and Jack Dongarra. Autotuning GEMM kernels for the Fermi GPU. *IEEE Transactions on Parallel and Distributed Systems*, 23(11):2045–2057, November 2012.

[146] Stanimire Tomov, Alan Ayala, Azzam Haidar, and Jack Dongarra. Fft-ecp api and high-performance library prototype for 2-d and 3-d ffts on large-scale heterogeneous systems with gpus. ECP WBS 2.3.3.13 Milestone Report FFT-ECP STML13-27, 2020-01 2020. revision 01-2020.

[147] Alan Ayala, Stanimire Tomov, Azzam Haidar, and Jack Dongarra. heffte: Highly efficient fft for exascale. In *International Conference on Computational Science (ICCS 2020)*, Amsterdam, Netherlands, 2020-06 2020.

[148] Hartwig Anzt, Terry Cojean, Yen-Chen Chen, Goran Flegar, Fritz Göbel, Thomas Grützmacher, Pratik Nayak, Tobias Ribizel, and Yu-Hsiang Tsai. Ginkgo: A high performance numerical linear algebra library. *Journal of Open Source Software*, 5(52):2260, 2020.

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

[150] Hartwig Anzt, Yen-Chen Chen, Terry Cojean, Jack Dongarra, Goran Flegar, Pratik Nayak, Enrique S. Quintana-Ortí, Yuhsiang M. Tsai, and Weichung Wang. Towards continuous benchmarking: An automated performance evaluation framework for high performance software. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '19, pages 9:1–9:11, New York, NY, USA, 2019. ACM.

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

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

[153] José I Aliaga, Hartwig Anzt, Thomas Grützmacher, Enrique S Quintana-Ortí, and Andrés E Tomás. Compressed basis gmres on high performance gpus. *arXiv preprint arXiv:2009.12101*, 2020.

[154] Yuhsiang M Tsai, Terry Cojean, Tobias Ribizel, and Hartwig Anzt. Preparing ginkgo for amd gpus–a testimonial on porting cuda code to hip. *arXiv preprint arXiv:2006.14290*, 2020.

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

[156] Jennifer A. Loe, Heidi K. Thornquist, and Erik G. Boman. *Polynomial Preconditioned GMRES in Trilinos: Practical Considerations for High-Performance Computing*, pages 35–45.

[157] Ahmad Abdelfattah, Hartwig Anzt, Aurelien Bouteiller, Anthony Danalis, Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, Stephen Wood, Panruo Wu, Ichitaro Yamazaki, and Asim YarKhan. SLATE working note 1: Roadmap for the development of a linear algebra library for exascale computing: SLATE: Software for linear algebra targeting exascale. Technical Report ICL-UT-17-02, Innovative Computing Laboratory, University of Tennessee, June 2017. revision 04-2018.

[158] Jakub Kurzak, Panruo Wu, Mark Gates, Ichitaro Yamazaki, Piotr Luszczek, Gerald Ragghianti, and Jack Dongarra. SLATE working note 3: Designing SLATE: Software for linear algebra targeting exascale. Technical Report ICL-UT-17-06, Innovative Computing Laboratory, University of Tennessee, September 2017. revision 09-2017.

[159] Mark Gates, Jakub Kurzak, Ali Charara, Asim YarKhan, and Jack Dongarra. SLATE: Design of a modern distributed and accelerated linear algebra library. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'19)*, Denver, CO, 2019. ACM.

[160] William F. Godoy, Norbert Podhorszki, Ruonan Wang, Chuck Atkins, Greg Eisenhauer, Junmin Gu, Philip Davis, Jong Choi, Kai Germaschewski, Kevin Huck, Axel Huebl, Mark Kim, James Kress, Tahsin Kurc, Qing Liu, Jeremy Logan, Kshitij Mehta, George Ostrouchov, Manish Parashar, Franz Poeschel, David Pugmire, Eric Suchyta, Keichi Takahashi, Nick Thompson, Seiji Tsutsumi, Lipeng Wan, Matthew Wolf, Kesheng Wu, and Scott Klasky. Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561, 2020.

[161] Qing Liu, Jeremy Logan, Yuan Tian, Hasan Abbasi, Norbert Podhorszki, Jong Youl Choi, Scott Klasky, Roselyne Tchoua, Jay Lofstead, Ron Oldfield, et al. Hello ADIOS: the challenges and lessons of developing leadership class i/o frameworks. *Concurrency and Computation: Practice and Experience*, 26(7):1453–1473, 2014.

[162] ADIOS2 documentation. https://adios2.readthedocs.io.

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

[164] James Ahrens, Kristi Brislawn, Ken Martin, Berk Geveci, C. Charles Law, and Michael Papka. Large-scale data visualization using parallel data streaming. *IEEE Computer Graphics and Applications*, 21(4):34–41, July/August 2001.

[165] Hank Childs, David Pugmire, Sean Ahern, Brad Whitlock, Mark Howison, Prabhat, Gunther H. Weber, and E. Wes Bethel. Extreme scaling of production visualization software on diverse architectures. *IEEE Computer Graphics and Applications*, 30(3):22–31, May/June 2010. DOI 10.1109/MCG.2010.51.

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

[167] Kenneth Moreland. Oh, $#*@! Exascale! The effect of emerging architectures on scientific discovery. In *2012 SC Companion (Proceedings of the Ultrascale Visualization Workshop)*, pages 224–231, November 2012. DOI 10.1109/SC.Companion.2012.38.

[168] Kenneth Moreland, Berk Geveci, Kwan-Liu Ma, and Robert Maynard. A classification of scientific visualization algorithms for massive threading. In *Proceedings of Ultrascale Visualization Workshop*, November 2013.

[169] Kenneth Moreland, Christopher Sewell, William Usher, Li ta Lo, Jeremy Meredith, David Pugmire, James Kress, Hendrik Schroots, Kwan-Liu Ma, Hank Childs, Matthew Larsen, Chun-Ming Chen, Robert Maynard, and Berk Geveci. VTK-m: Accelerating the visualization toolkit for massively threaded architectures. *IEEE Computer Graphics and Applications*, 36(3):48–58, May/June 2016. DOI 10.1109/MCG.2016.48.

[170] Kenneth Moreland. The vtk-m user's guide. techreport SAND 2018-0475 B, Sandia National Laboratories, 2018. http://m.vtk.org/images/c/c8/VTKmUsersGuide.pdf.

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

[172] Li-ta Lo, Chris Sewell, and James Ahrens. PISTON: A portable cross-platform framework for data-parallel visualization operators. In *Eurographics Symposium on Parallel Graphics and Visualization*, 2012. DOI 10.2312/EGPGV/EGPGV12/011-020.

[173] Matthew Larsen, Jeremy S. Meredith, Paul A. Navrátil, and Hank Childs. Ray tracing within a data parallel framework. In *IEEE Pacific Visualization Symposium (PacificVis)*, April 2015. DOI 10.1109/PACIFICVIS.2015.7156388.

[174] Kenneth Moreland, Matthew Larsen, and Hank Childs. Visualization for exascale: Portable performance is critical. In *Supercomputing Frontiers and Innovations*, volume 2, 2015. DOI 10.2312/pgv.20141083.

[175] H. Carter Edwards, Daniel Sunderland, Chris Amsler, and Sam Mish. Multicore/GPGPU portable computational kernels via multidimensional arrays. In *IEEE Cluster*, September 2011.

[176] Jeremy S. Meredith, Sean Ahern, Dave Pugmire, and Robert Sisneros. EAVL: The extreme-scale analysis and visualization library. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 21–30, 2012. DOI 10.2312/EGPGV/EGPGV12/021-030.

[177] Kenneth Moreland, Brad King, Robert Maynard, and Kwan-Liu Ma. Flexible analysis software for emerging architectures. In *2012 SC Companion (Petascale Data Analytics: Challenges and Opportunities)*, pages 821–826, November 2012. DOI 10.1109/SC.Companion.2012.115.

[178] William Schroeder, Robert Maynard, and Berk Geveci. Flying edges: A high-performance scalable isocontouring algorithm. In *Large Data Analysis and Visualization (LDAV)*, October 2015.

[179] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. The Spack package manager: Bringing order to HPC software chaos. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2015.

[180] John K. Salmon, Mark A. Moraes, Ron O. Dror, and David E. Shaw. Parallel random numbers: As easy as 1, 2, 3. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, November 2011.

[181] Tonmoy Dey, Kento Sato, Bogdan Nicolae, Jian Guo, Jens Domke, Weikuan Yu, Franck Cappello, and Kathryn Mohror. Optimizing asynchronous multi-level checkpoint/restart configurations with machine learning. In *HPS'20: The 2020 IEEE International Workshop on High-Performance Storage*, New Orleans, USA, 2020.

[182] Bogdan Nicolae, Jiali Li, Justin Wozniak, George Bosilca, Matthieu Dorier, and Franck Cappello. Deepfreeze: Towards scalable asynchronous checkpointing of deep learning models. In *CGrid'20: 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing*, pages 172–181, Melbourne, Australia, 2020.

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

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

[185] Hank Childs, Eric Brugger, Brad Whitlock, Jeremy Meredith, Sean Ahern, David Pugmire, Kathleen Biagas, Mark Miller, Cyrus Harrison, Gunther H. Weber, Hari Krishnan, Thomas Fogal, Allen Sanderson, Christoph Garth, E. Wes Bethel, David Camp, Oliver Rübel, Marc Durant, Jean M. Favre, and Paul Navrátil. VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data. In *High Performance Visualization—Enabling Extreme-Scale Scientific Insight*, pages 357–372. CRC Press/Francis–Taylor Group, October 2012.

[186] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. The ALPINE In Situ Infrastructure: Ascending from the Ashes of Strawman. In *Proceedings of the Third Workshop of In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV), held in conjunction with SC17*, Denver, CO, USA, October 2017.

[187] Utkarsh Ayachit, Andrew Bauer, Berk Geveci, Patrick O'Leary, Kenneth Moreland, Nathan Fabian, and Jeffrey Mauldin. Paraview catalyst: Enabling in situ data analysis and visualization. In *Proceedings of the First Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV 2015)*, pages 25–29, November 2015.

[188] Hamish A. Carr, Gunther H. Weber, Christopher M. Sewell, Oliver Rübel, Patricia Fasel, and James P. Ahrens. Scalable contour tree computation by data parallel peak pruning. *Transactions on Visualization and Computer Graphics*, 2019. In press.

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

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

[191] Qun Liu, Subhashis Hazarika, John M. Patchett, James P. Ahrens, and Ayan Biswas. Poster: Deep learning-based feature-aware data modeling for complex physics simulations. In *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 11 2019. To Appear in SC '19.

[192] S. Dutta, J. Woodring, Han-Wei Shen, J. Chen, and J. Ahrens. Homogeneity guided probabilistic data summaries for analysis and visualization of large-scale data sets. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pages 111–120, 2017.

[193] Soumya Dutta and Han-Wei Shen. Distribution driven extraction and tracking of features for time-varying data analysis. *IEEE Trans. on Vis. and Comp. Graphics*, 22(1):837–846, 2016.

[194] Aaditya G Landge, Valerio Pascucci, Attila Gyulassy, Janine C Bennett, Hemanth Kolla, Jacqueline Chen, and Peer-Timo Bremer. In-situ feature extraction of large scale combustion simulations using segmented merge trees. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1020–1031. IEEE, 2014.

[195] Steve Petruzza, Sean Treichler, Valerio Pascucci, and Peer-Timo Bremer. Babelflow: An embedded domain specific language for parallel analysis and visualization. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 463–473. IEEE, 2018.

[196] Xavier Bonaventura, Miquel Feixas, Mateu Sbert, Lewis Chuang, and Christian Wallraven. A survey of viewpoint selection methods for polygonal models. *Entropy*, 20(5), 2018.

[197] Nicole Marsaglia. Automatic camera selection for in situ visualization. Technical Report AREA-202001-Marsaglia, University of Oregon, Computer and Information Sciences Department, 1 2020. Available at https://www.cs.uoregon.edu/Reports/AREA-202001-Marsaglia.pdf.

[198] Sudhanshu Sane, Roxana Bujack, and Hank Childs. Revisiting the Evaluation of In Situ Lagrangian Analysis. In Hank Childs and Fernando Cucchietti, editors, *Eurographics Symposium on Parallel Graphics and Visualization*. The Eurographics Association, 2018.

[199] Sudhanshu Sane, Hank Childs, and Roxana Bujack. An Interpolation Scheme for VDVP Lagrangian Basis Flows. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, pages 109–118, Porto, Portugal, June 2019.

[200] Roba Binyahib, David Pugmire, Boyana Norris, and Hank Childs. A Lifeline-Based Approach for Work Requesting and Parallel Particle Advection. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, Vancouver, Canada, October 2019.

[201] Lennart Noordsij, Steven van der Vlugt, Mohamed Bamakhrama, Zaid Al-Ars, and Peter Lindstrom. Parallelization of variable rate decompression through metadata. In *Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 245–252, Mar 2020.

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

[203] Stephen Hamilton, Randal Burns, Charles Meneveau, Perry Johnson, Peter Lindstrom, John Patchett, and Alexander Szalay. Extreme event analysis in next generation simulation architectures. In *ISC High Performance 2017*, pages 277–293, 2017.

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

[205] James Diffenderfer, Alyson Fox, Jeffrey Hittinger, Geoffrey Sanders, and Peter Lindstrom. Error analysis of ZFP compression for floating-point data. *SIAM Journal on Scientific Computing*, 41(3):A1867–A1898, 2019.

[206] Dorit Hammerling, Allison Baker, Alexander Pinard, and Peter Lindstrom. A collaborative effort to improve lossy compression methods for climate data. In *5th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-5)*, Nov 2019.

[207] Duong Hoang, Brian Summa, Harsh Bhatia, Peter Lindstrom, Pavol Klacansky, Will Usher, Peer-Timo Bremer, and Valerio Pascucci. Efficient and flexible hierarchical data layouts for a unified encoding of scalar field precision and resolution. *IEEE Transactions on Visualization and Computer Graphics*, 2020. To appear.

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

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

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

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

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

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

[214] Franck Cappello, Peter Lindstrom, and Sheng Di. Lossy compression for scientific data. IEEE/ACM SC 2020 Tutorials, 2020.

[215] Franck Cappello, Peter Lindstrom, Sheng Di, Jon Calhoun, Pascal Grosset, Katrin Heitmann, and Allison Baker. Lossy compression for scientific data: Success stories. IEEE/ACM SC 2020 Panels, 2020.

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

[217] E4S Validation Test Suite. https://github.com/E4S-Project/testsuite.

[218] R Priedhorsky and TC Randles. Charliecloud: Unprivileged containers for user-defined software stacks in hpc. Technical Report LA-UR-16-22370, Los Alamos National Laboratory, 2016. available as http://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-16-22370.

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

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

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

[222] Bauer MW Kurtzer GM, Sochat V. Singularity: Scientific containers for mobility of compute. *PLoS ONE*, may 2017. available as https://doi.org/10.1371/journal.pone.0177459.

[223] Tao B. Schardl, William S. Moses, and Charles E. Leiserson. Tapir: Embedding fork-join parallelism into llvm's intermediate representation. *SIGPLAN Not.*, 52(8):249–265, January 2017.

[224] J. Ahrens, S. Jourdain, P. OLeary, J. Patchett, D. H. Rogers, and M. Petersen. An image-based approach to extreme scale in situ visualization and analysis. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 424–434, Nov 2014.

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

[226] Todd Gamblin, Gregory Becker, Massimiliano Culpo, Mario Melara, Peter Scheibel, and Adam J. Stewart. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2019*, Denver, CO, November 18 2019. Full day.

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

[228] Todd Gamblin and Gregory Becker. Spack Tutorial. In *1st Workshop on NSF and DOE High Performance Computing Tools*, Eugene, OR, July 10-11 2019. University of Oregon.

[229] Levi Baber, Adam J. Stewart, Gregory Becker, and Todd Gamblin. Tutorial: Managing HPC Software Complexity with Spack. In *Practice and Experience in Advanced Research Computing (PEARC'19)*, Chicago, IL, July 31 2019. Half day.

[230] Todd Gamblin, Gregory Becker, Massimiliano Culpo, and Michael Kühn. Tutorial: Managing HPC Software Complexity with Spack. In *ISC High Performance*, Houston, TX, June 16 2019. Half day.

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

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

[233] Todd Gamblin, Adam Stewart, Johannes Albert von der Gönna an Marc Pérache, and Matt Belhorn. Spack Community Birds-of-a-Feather Session. In *Supercomputing 2018*, Dallas, TX, November 13 2018.

[234] Todd Gamblin, Gregory Becker, Massimiliano Culpo, Gregory L. Lee, Matt Legendre, Mario Melara, and Adam J. Stewart. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2018*, Dallas, TX, November 12 2018. Full day.

[235] Todd Gamblin, William Scullin, Matt Belhorn, Mario Melara, and Gerald Ragghianti. Spack State of the Union. In *Exascale Computing Project 2nd Annual Meeting*, Knoxville, TN, February 6-8 2018.

[236] Todd Gamblin, Gregory Becker, Massimiliano Culpo, Gregory L. Lee, Matt Legendre, Mario Melara, and Adam J. Stewart. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2017*, Salt Lake City, Utah, November 13 2017. Full day.

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

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

[239] Todd Gamblin, Massimiliano Culpo, Gregory Becker, Matt Legendre, Greg Lee, Elizabeth Fischer, and Benedikt Hegner. Tutorial: Managing HPC Software Complexity with Spack. In *Supercomputing 2016*, Salt Lake City, Utah, November 13 2016. Half day.

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

[241] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny V. Dobrev, Y. Dudouit, A. Fisher, Tz. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, and S. Zampini. MFEM: A modular finite element library. *Computers & Mathematics with Applications*, 2020.

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

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

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

[245] R. W. Anderson, V. A. Dobrev, T. V. Kolev, R. N. Rieben, and V. Z. Tomov. High-order multi-material ALE hydrodynamics. *SIAM J. Sc. Comp.*, 40(1):B32–B58, 2018.

[246] V. A. Dobrev, T. V. Kolev, D. Kuzmin, R. N. Rieben, and V. Z. Tomov. Sequential limiting in continuous and discontinuous Galerkin methods for the Euler equations. *J. Comput. Phys.*, 356:372–390, 2018.

[247] R. W. Anderson, V. A. Dobrev, T. V. Kolev, D. Kuzmin, M. Quezada de Luna, R. N. Rieben, and V. Z. Tomov. High-order local maximum principle preserving (MPP) discontinuous Galerkin finite element method for the transport equation. *J. Comput. Phys.*, 334:102–124, 2017.

[248] V. A. Dobrev, T. V. Kolev, R. N. Rieben, and V. Z. Tomov. Multi-material closure model for high-order finite element Lagrangian hydrodynamics. *Int. J. Numer. Meth. Fluids*, 82(10):689–706, 2016.

[249] V. A. Dobrev, T. V. Kolev, and R. N. Rieben. High order curvilinear finite elements for elastic–plastic Lagrangian dynamics. *J. Comput. Phys.*, 257, Part B:1062 – 1080, 2014.

[250] V. A. Dobrev, T. E. Ellis, Tz. V. Kolev, and R. N. Rieben. High-order curvilinear finite elements for axisymmetric Lagrangian hydrodynamics. *Computers and Fluids*, 83:58–69, 2013.

[251] V. A. Dobrev, T. V. Kolev, and R. N. Rieben. High-order curvilinear finite element methods for Lagrangian hydrodynamics. *SIAM J. Sc. Comp.*, 34(5):B606–B641, 2012.

[252] V. A. Dobrev, T. E. Ellis, Tz. V. Kolev, and R. N. Rieben. Curvilinear finite elements for Lagrangian hydrodynamics. *Int. J. Numer. Meth. Fluids*, 65(11-12):1295–1310, 2011.

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