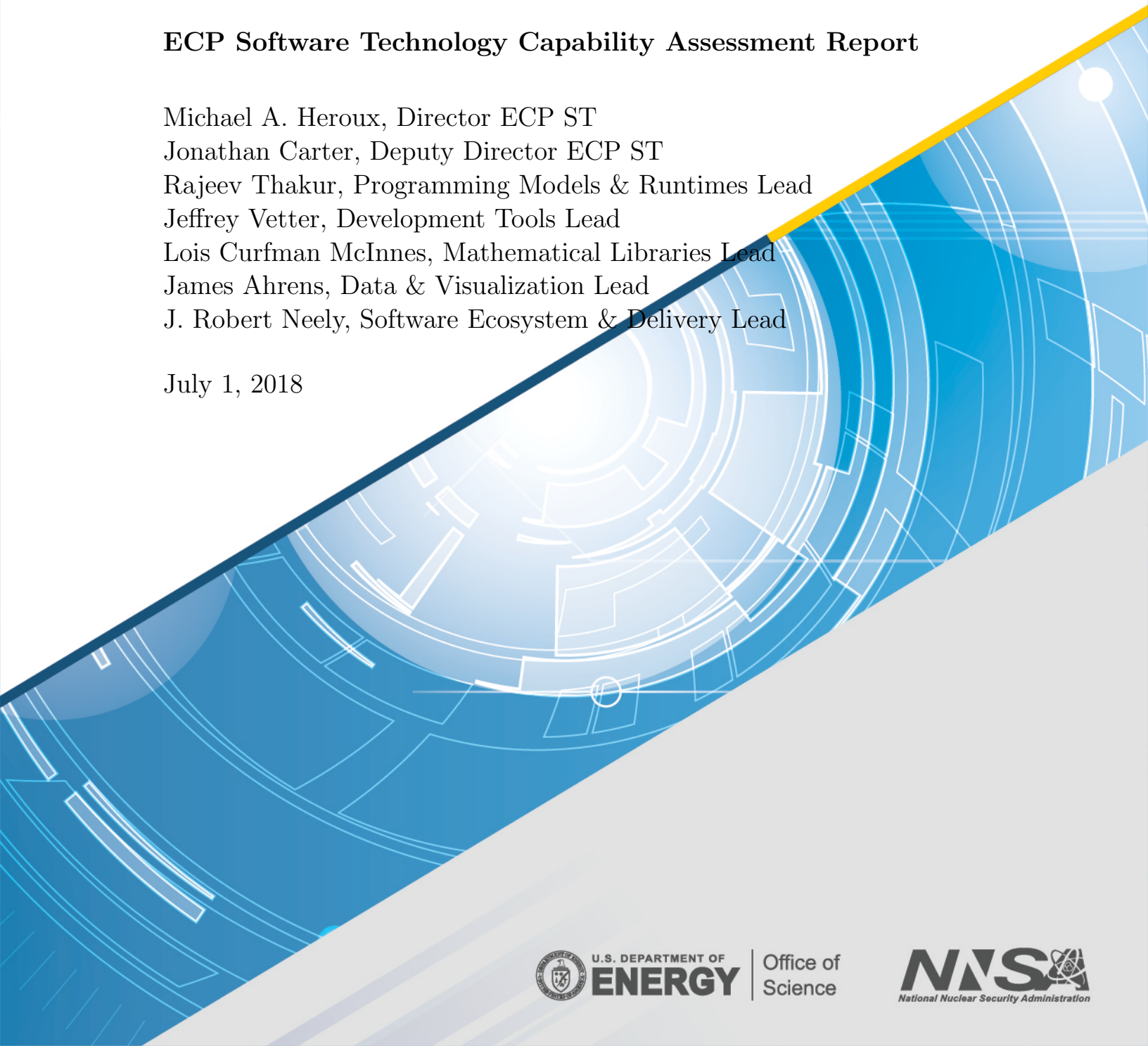
EXASCALE COMPUTING PROJECT

ECP-RPT-ST-0001-2018

ECP Software Technology Capability Assessment Report

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

July 1, 2018



U.S. DEPARTMENT OF ENERGY | Office of Science

NNSA National Nuclear Security Administration

ECP-RPT-ST-0001-2018

ECP Software Technology Capability Assessment Report

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

July 1, 2018

REVISION LOG

Version	Date	Description
1.0	July 1, 2018	*ECP ST Capability Assessment Report*

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 (targeting approximately every six months). Highlights from the report are presented here.

Origin of the ECP ST CAR: The CAR is a follow-on and expansion of the *ECP Software Technology Gap Analysis* [1]. The CAR includes a gap analysis as well as a broader description of ECP ST efforts, capabilities and plans.

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 89 software products (Section 3.1) in five technical areas (Table 1). 33 of the 89 products are broadly used in the HPC community and require substantial investment and transformation in preparation for Exascale architectures. An additional 23 are important to some existing applications and typically represent new capabilities that enable new usage models for realizing the potential that Exascale platforms promise. The remaining products are in early development phases, addressing emerging challenges and opportunities that Exascale platforms present.

Programming Models & Runtimes: ECP ST is developing key enhancements to MPI and OpenMP, addressing in particular the important design and implementation challenges of combining massive inter-node and intra-node concurrency into an application. We are also developing a diverse collection of products that further address next generation node architectures, to improve realized performance, ease of expression and performance portability.

Development Tools: We are enhancing existing widely used performance tools and developing new tools for next-generation platforms. As node architectures become more complicated and concurrency even more necessary, 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.

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. 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 1.2.1).

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.

SW Ecosystem & Delivery: This new technical area of ECP ST provides important enabling technologies such as containers and experimental OS environments that allow ECP ST to provide requirements, analysis and design input for vendor products. This area also provides the critical resources and staffing that will enable ECP ST to perform continuous integration testings, 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.

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

ECP ST Project Restructuring: ECP ST completed a significant restructuring in November 2017 (Section 1.3). We reduced the number of technical areas from 8 to 5 and reduced the number of L4 projects significantly by simplifying the organization of ATDM projects. We introduced new projects for software development kits that are a key organizational feature for designing, testing and delivering our software. Finally, we introduced a new technical area (SW Ecosystem & Delivery) that provides the critical capabilities we need for delivering a sustainable software ecosystem.

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 impact goals/metrics, 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 55 projects and more than 250 contributors to create the ECP ST software stack.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The Exascale Computing Project Software Technology (ECP ST) focus area represents the key bridge between Exascale systems and the scientists developing applications that will run on those platforms. ECP offers a unique opportunity to build a coherent set of software (often referred to as the "software stack") that will allow application developers to maximize their ability to write highly parallel applications, targeting multiple Exascale architectures with runtime environments that will provide high performance and resilience. But applications are only useful if they can provide scientific insight, and the unprecedented data produced by these applications require a complete analysis workflow that includes new technology to scalably collect, reduce, organize, curate, and analyze the data into actionable decisions. This requires approaching scientific computing in a holistic manner, encompassing the entire user workflow—from conception of a problem, setting up the problem with validated inputs, performing high-fidelity simulations, to the application of uncertainty quantification to the final analysis. The software stack plan defined here aims to address all of these needs by extending current technologies to Exascale where possible, by performing the research required to conceive of new approaches necessary to address unique problems where current approaches will not suffice, and by deploying high-quality and robust software products on the platforms developed in the Exascale systems project. The ECP ST portfolio has established a set of interdependent projects that will allow for the research, development, and deployment 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 runtime, mathematical libraries and frameworks, tools, 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 five projects funded by the DOE Office of Science (ASCR) that were selected in October 2016 via an RFI and RFP process, considering prioritized requirements.

2. A similar number of ongoing DOE NNSA/ASC funded projects that are part of the Advanced Technology Development and Mitigation (ATDM) program, which is in its fourth 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 1.3.

1.1 BACKGROUND

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

Mathematical libraries and tools for supercomputers have traditionally been developed at DOE laboratories and universities and ported to the new computer architectures when they are deployed. These math libraries and tools have been remarkably robust and have supplied some of the most impactful improvements in application performance and productivity. The challenges have been the constant adapting and tuning to rapidly changing architectures.

Programming paradigms and the associated programming environments that include compilers, debuggers, message passing, and associated runtimes have traditionally been developed by vendors, DOE laboratories, and universities. The same can be said for file system and storage software. An observation is that the vendor is ultimately responsible for providing a programming environment and file system with the supercomputer, but there is often a struggle to get the vendors to support software developed by others or to invest in new ideas that have few or no users yet. Another observation is that file-system software plays a key role in overall system resilience, and the difficulty of making the file-system software resilient has grown nonlinearly with the scale and complexity of the supercomputers.

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.
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	A unified set of robust, lower-level software libraries as well as end-user tools that help address the complexities of developing higher-level software and leveraging and utilizing Exascale system components and resources. Programming tools, libraries, and system support for incorporating resilience into application codes that enables them to run successfully and efficiently in the presence of faults experienced on the system. Oversight of development across software technology to ensure the teams are communicating and coordinating, other focus areas are included in the execution, interfaces are agreed upon and standardized where necessary, and interdependencies across projects are effectively managed.

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

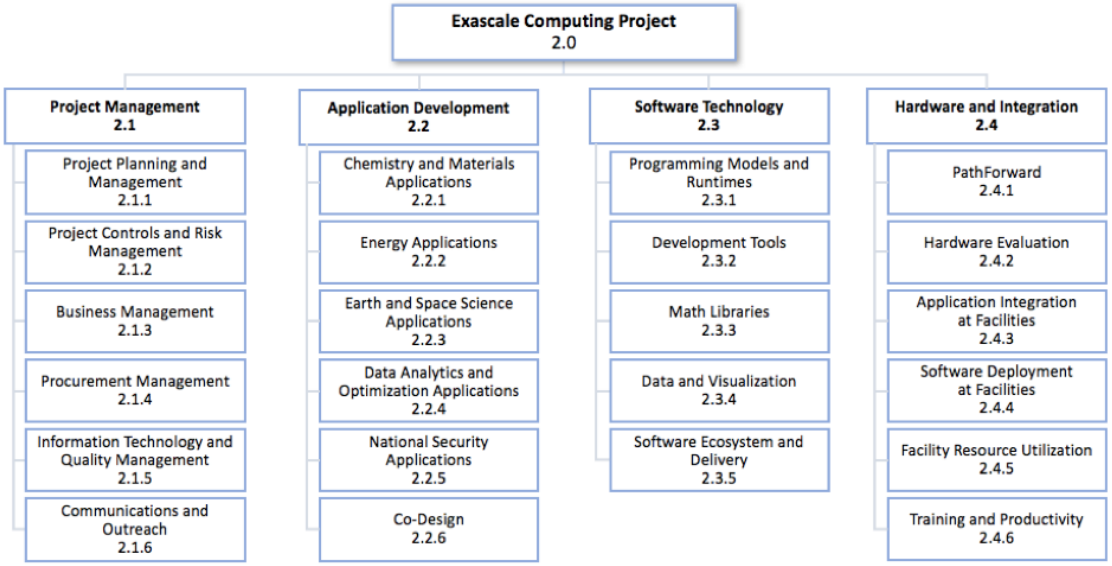


Figure 1: The ECP Work Breakdown Structure through Level 3 (L3).

In addition to the lessons learned from the 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 via a combination of tasks, threads and vectorization, and more than one hundred thousand nodes. 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.

- **Resilience:** As hardware resilience decreases due to the number of components and reduced voltage, software resilience must be developed to take up the slack and allow the Exascale system to be adaptable to component failures without the entire system crashing. Initial concerns about resilience at the start of Exascale efforts have diminished, and the availability of non-volatile memory should dramatically improve checkpoint/restart performance. Even so, we need to keep a focus on this issue.

- **Power consumption:** Exascale systems have been given an aggressive power consumption goal of 20-30 MW, not much more than the power consumed by the largest systems of today. Meeting this goal will require the development of power monitoring and management software that does not exist today.

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. Traditionally, there has been a focus on just one new supercomputer every couple of years. ECP has a goal of enabling at least two diverse architectures, and the ECP-developed software needs to be able to run efficiently on all of them. Some code divergence is inevitable, but careful software design, and the use of performance portability layers can minimize the amount of code targeted at a specific platform.
- **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 that does not exist today.

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

1.2 ECP SOFTWARE TECHNOLOGY APPROACH

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

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

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.

1.2.1 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:

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 [2]

.

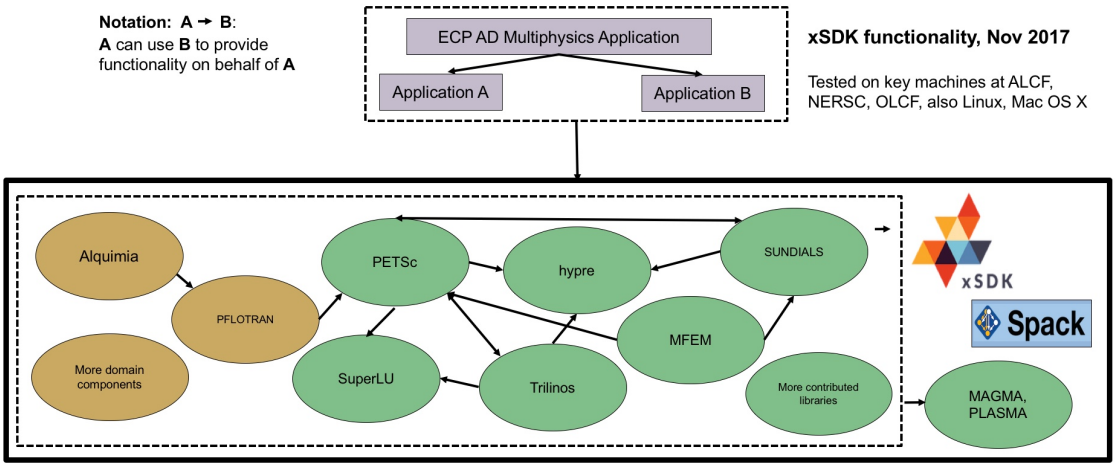


Figure 2: The xSDK is the first SDK for ECP ST, in the Mathematical Libraries technical area 1. The xSDK provides the collaboration environment for improving build, install and testing capabilities for member packages such as hypre, PETSc, SuperLU and Trilinos (and other products with green background). Domain components (see orange ovals) are also an important category of the ecosystem, providing leveraged investments for common components in a specific scientific software domain. xSDK capabilities are essential for supporting the multi-physics and multi-scale application requirement that lead to combined use of xSDK libraries. Furthermore, the availability of advanced software platforms such as GitHub, Confluence, JIRA and others enable the level of collaboration needed to create an SDK from independently developed packages.

ECP ST SDKs As part of the delivery of ECP ST capabilities, we will establish and grow a collection of SDKs. The new layer of aggregation that SDKs represent are important for improving all aspects of product development and delivery. The communities that will emerge from SDK efforts will lead to better collaboration and higher quality products. Established community policies will provide a means to grow SDKs beyond ECP to include any relevant external effort. The meta-build systems (based on Spack) will play an important role in managing the complexity of building the ECP ST software stack, by providing a new layer where versioning, consistency and build options management can be addressed at a mid-scope, below the global build of ECP ST products. Each ECP ST L3 (five of them) has funds for an SDK project from which we will identify and establish at least one SDK effort. Fortunately, we will be able to leverage an existing SDK in the Math Libraries sub-element to inform our broader efforts. This SDK, called the xSDK, has been in existence for several years and has proven the value of an SDK approach in its domain (Figure 2).

The xSDK Initially funded by the DOE Office of Advanced Scientific Computing Research and the office of Biological and Environmental Research as part of the IDEAS Project [3], the xSDK is a collection of independent math library packages, initially the popular libraries hypre, PETSc, SuperLU and Trilinos. The xSDK was established in recognition that collaboration across independent library development efforts could have a tremendous positive impact on the math libraries capabilities provided to users, and the productivity of library developers and sustainability of library software. Figure 2 illustrates the scope and interaction of xSDK packages and Figure 3 lists the community policies that govern xSDK activities and set expectations for future xSDK members. While we recognize that xSDK experiences cannot be blindly applied to creation of new SDKs in ECP, the xSDK does provide a concrete, working example to guide ECP ST SDK efforts going forward.

1.2.2 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,

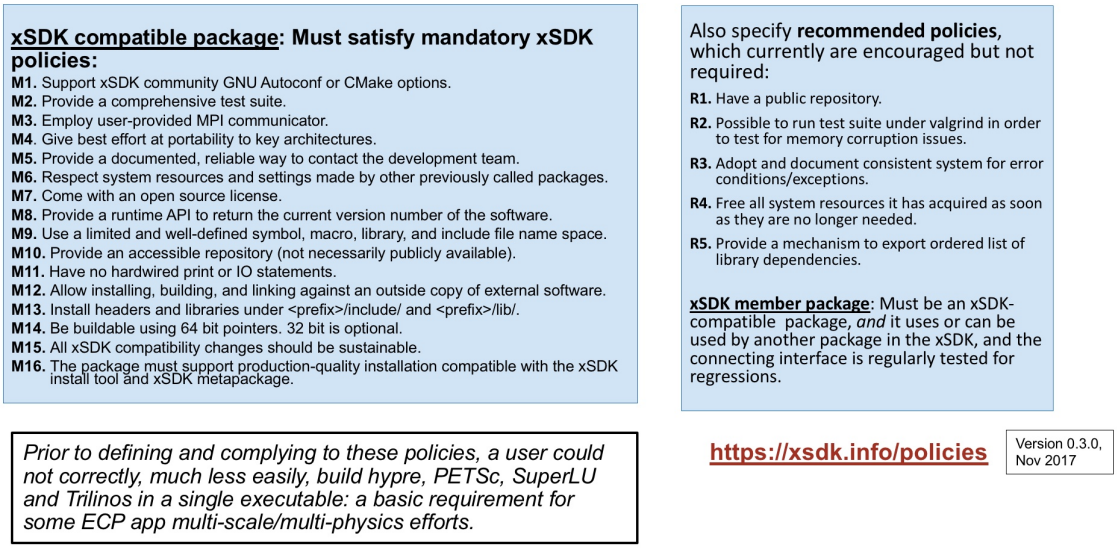


Figure 3: xSDK Community Policies emerged from challenging and passionate discussions about essential values of the math libraries community. Once established, these community policies represent a living statement of what it means to be part of an SDK, and are used as the criteria for welcoming future members.

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 4.

Users access ECP ST products using these basic mechanisms (see Figure 9 for deliverable statistics):

- **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 [4]. 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 4.5.8) and LLVM (Section 4.1.14), and parallel programming environments such as MPICH (Section 4.1.8), OpenMPI (Section 4.1.12) and OpenMP (Section 4.1.16). Development tools such as PAPI (Section 4.2.6) and TAU (Section 4.2.10) are often part of this suite, if not already included in the vendor stack. Math and data libraries such as PETSc (Section 4.3.7), Trilinos (Section 4.3.5), HDF5 (Section 4.4.9) 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) [5] that are focused on improving uniformity of software stacks across facilities.

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

- **Binary distributions:** Approximately 10 ECP ST products are available via binary distributions such as common Linux distributions, in particular via OpenHPC[6]. 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.

1.3 ECP ST PROJECT RESTRUCTURING

The initial organization of ECP ST was based on discussions that occurred over several years of Exascale planning within DOE, especially the DOE Office of Advanced Scientific Computing Research (ASCR). Figure 6 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 (presently 55 due to further merging in SW Ecosystem & Delivery). Figure

1.4 NEW PROJECT EFFORTS

ECP ST has initiated four newly-funded efforts as a result of findings in our September 2017 Gap Analysis [1]. We anticipate several additional efforts in the coming months in order to mitigate some of the higher risks we

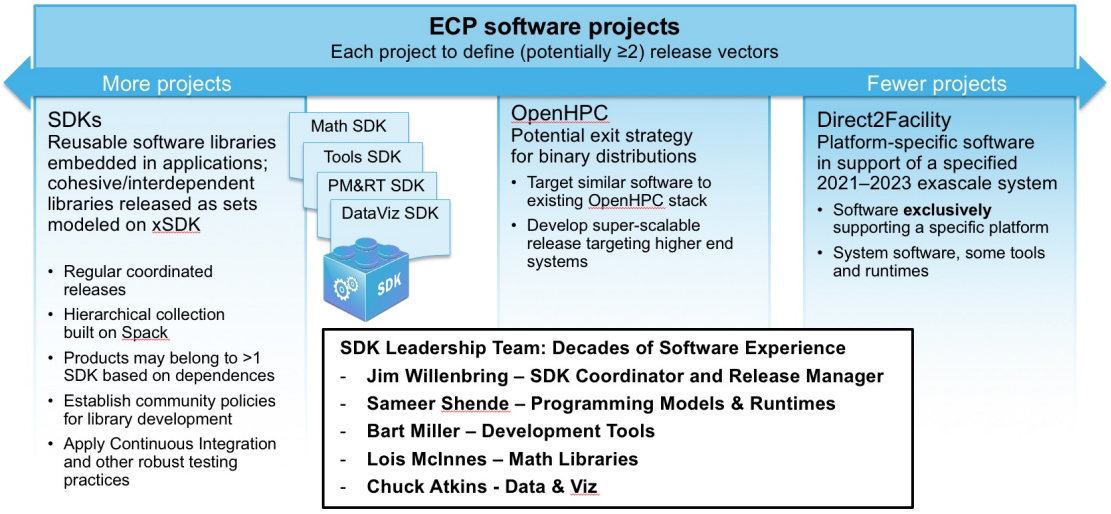


Figure 4: The ECP ST software stack is delivered to the user community through several channels. Key channels are via source code, increasing using SDKs, direct to Facilities in collaboration with ECP HI, via binary distributions, in particular the OpenHPC project and via HPC vendors. The SDK leadership team includes ECP ST team members with decades of experience delivering scientific software products.

- Phase 1: 66 total projects
  - 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: 56 total projects (now 55 after further merging in 2.3.5)
  - 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 mapped 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 porfolio without disruption to the ECP-ATDM mapping.

Figure 5: Project remapping summary from Phase 1 (through November 2017) to Phase 2 (After November 2017)

WBS	Role/Area	Leader	Transition
1.3	ECP ST Director	Rajeev Thakur	Renumbered to 2.3. Thakur left director role, continues as lead of 2.3.1 Programming Models & Runtimes. Mike Heroux new director.
1.3	ECP ST Deputy Director	Pat McCormick	McCormick left deputy role, continued as PI of 2.3.1.08 Legion project. Jonathan Carter new deputy director.
1.3.1	Programming Models & Runtimes	Rajeev Thakur	Renumbered to 2.3.1, renamed to Programming Models & Runtimes, otherwise unchanged.
1.3.2	Tools	Jeff Vetter	Renumbered to 2.3.2, renamed to Development Tools, otherwise unchanged.
1.3.3	Math/Scientific Libs	Mike Heroux	New leader Lois Curfman McInnes, renamed Mathematical Libraries, new number 2.3.3.
1.3.4	Data Management & Workflows	Rob Ross	Combined with 1.3.5 to create 2.3.4. Jim Ahrens leader.
1.3.5	Data Analytics & Visualization	Jim Ahrens	Combined with 1.3.4 to create 2.3.4. Jim Ahrens leader.
1.3.6	System Software	Martin Schulz	Combined with 1.3.7 and 1.3.8 into 2.3.5. Rob Neely leader.
1.3.7	Resilience	Al Geist	Combined with 1.3.6 and 1.3.8 into 2.3.5. Rob Neely leader.
1.3.8	Integration	Rob Neely	Combined with 1.3.6 and 1.3.7 into 2.3.5. Rob Neely leader.

Table 3: ECP ST technical areas were reduced from 8 to 5 in November 2017. This figure shows how areas were remapped and merged. In addition, the ECP ST Director and Deputy Director changed from Rajeev Thakur (who continues as the Programming Models & Runtimes lead) and Pat McCormick to Mike Heroux and Jonathan Carter, respectively.

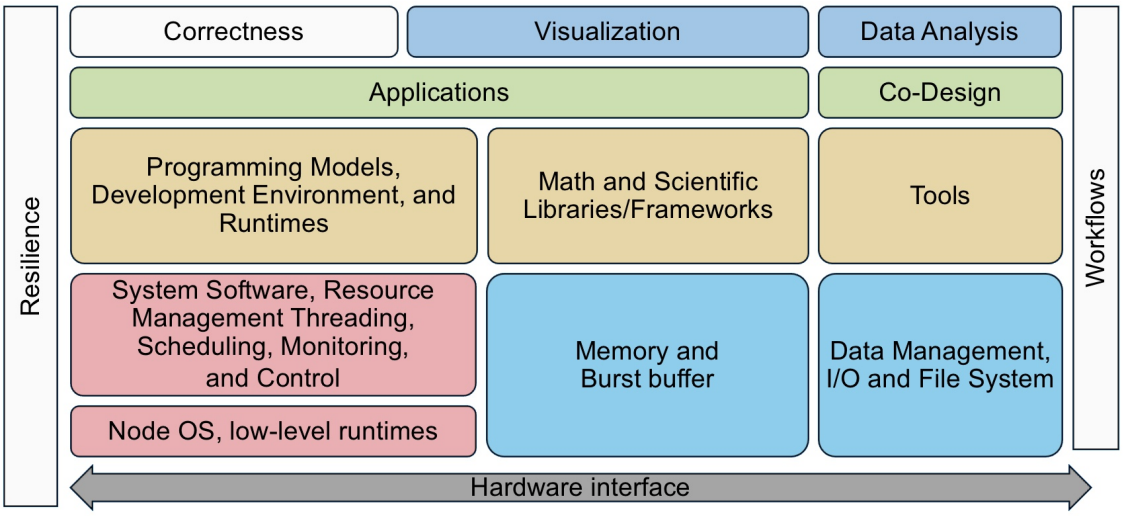


Figure 6: ECP ST before November 2017 reorganization. This conceptually layout emerged from several years of Exascale planning, conducted primarily within the DOE Office of Advanced Scientific Computing Research (ASCR). After a significant restructuring of ECP that removed much of the facilities activities 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.

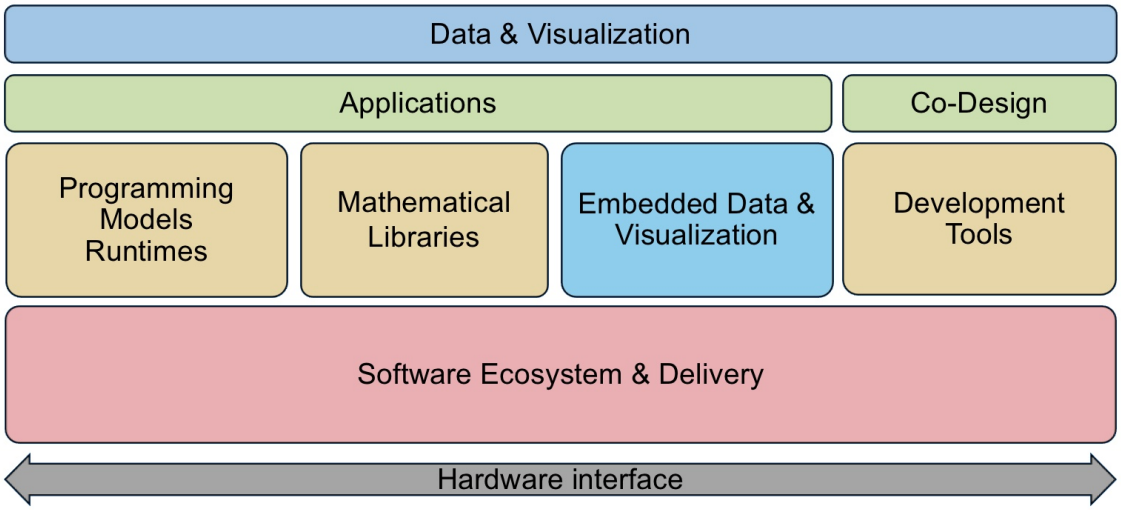


Figure 7: ECP ST after November 2017 reorganization. This diagram more accurately reflects the priorities and efforts of ECP ST given the new ECP project scope and the demands that we foresee.

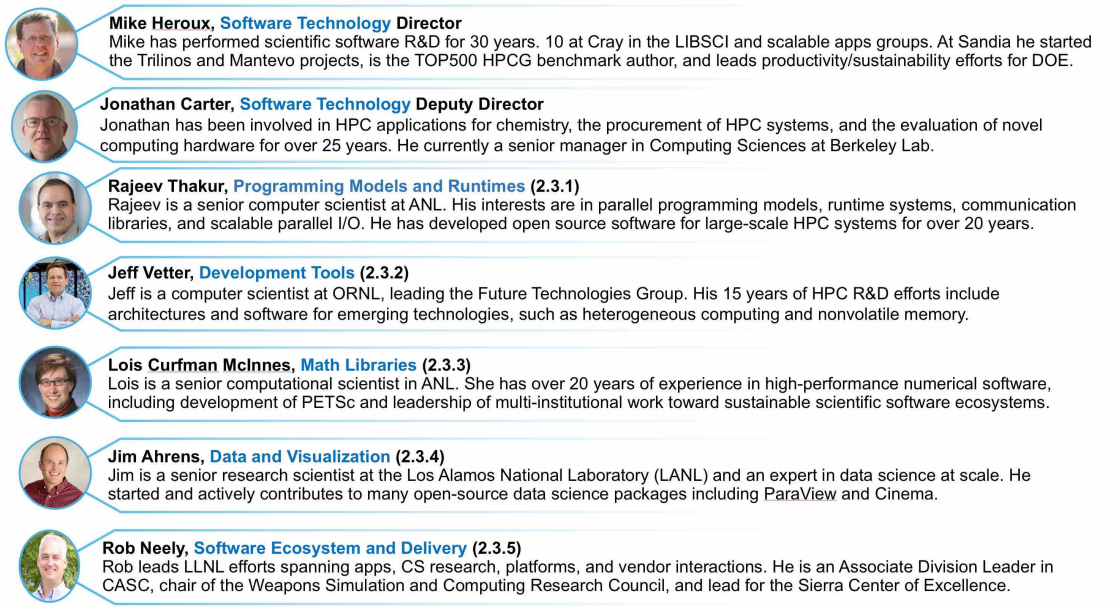


Figure 8: ECP ST Leadership Team as of November 2017.

have identified in programming environments and data management. We will report this second phase of additions in the next version of this report.

1.4.1 FFTs

ECP ST has initiated two new efforts in Fast Fourier Transforms (FFTs). FFTs provide an essential mathematical tool to many application areas. From the very beginning of HPC, vendors have provided optimized FFT libraries for their users. The advent of FFTW [7], a tunable high-performance library with a well-designed interface, enabled a *de facto* standardization of FFT interfaces, and an effective source base for vendor libraries, which adapted FFTW source for their platforms.

There is some concern in the community that FFTW is no longer actively developed, nor well prepared for emerging platforms. Furthermore, FFTW's strong copylefting license (which forces its users to make their own software open source in the same way) has always been a challenge to users. While vendors are still committed to providing optimized FFT libraries, whether or not FFTW is available, we believe it is prudent to explore a new software stack and have funded a short-term project to explore this possibility. The new library will also explore problem formulations that could significantly reduce the computational cost of FFTs. This new effort, called FFTX, will be led by Lawrence Berkeley National Lab, under the existing math libraries project (Section 4.3.8).

A second FFT project will address a consensus opportunity to provide a sustainable 3D FFT library built from established but *ad hoc* software tools that have traditionally been part of application codes, but not extracted as independent, supported libraries. These 3D FFTs rely on third-party 1D FFTs, either from FFTW or from vendor libraries.

The goal of this second project, FFT-ECP, led by the University of Tennessee and integrated into one of its existing projects (Section 4.3.10) is to:

- Collect existing FFT capabilities recently made available from ECP application teams (LAMMPS/fftMPI and HACC/SWFFT).
- Assess gaps and make available as a sustainable math library.
- Explore opportunities to build 3D FFT libraries on vendor 1D and 2D kernels, especially leveraging on-node concurrency from 2D and batched 1D formulations.
- Focus on capabilities for Exascale platforms.

- Emphasize leverage of vendor capabilities and addressing vendor deficiencies over creation of new and independent software stack.

This effort, while not addressing the concerns about FFTW directly, is essential to providing a new and sustainable FFT software stack that leverages the large investment by the broader HPC community in FFT software. The payoff from this effort is almost guaranteed. Also, should the FFTX project also go forward, it will provide an FFTW-compatible interface that would allow FFT-ECP to use FFTX as one option, in addition to external FFT libraries.

1.4.2 LLNL Math Libraries

When ECP ST started, some important capabilities were not part of the original portfolio, even though their engagement is essential for ECP application success. This is true of the LLNL math library *hypre* [8]. This library is widely used to provide scalable multigrid preconditioners across several ECP applications. Funding to support adaptation of *hypre* in preparation for Exascale platforms at science lab facilities, e.g., Argonne, and for ECP science applications was not part of the original ECP ST portfolio. We have added funding for this effort, starting in June 2018. In addition, we provided new funding for another LLNL math library, MFEM [9], so that the MFEM team can participate in SDK efforts for math libraries.

2. ECP ST TECHNICAL AREAS

2.1 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.

2.1.1 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.

2.1.2 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.

2.1.3 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.

2.1.4 Plan

PMR contains fifteen L4 projects. To ensure relevance to DOE missions, these efforts leverage and collaborate with existing activities within the broader HPC community. Initial efforts focus 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 provide demonstrations of parallel performance of algorithms on pre-exascale, leadership-class machines–at first on test problems, but eventually in actual applications (which will require close collaboration with the AD and HI teams). Initial efforts also inform research into exascale-specific algorithms and requirements that will be implemented across the software stack. The supported projects target and implement 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. 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.

2.1.5 Risks and Mitigation Strategies

The mainstream activities of PMR focus on advancing the capabilities of the Message Passing Interface (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 both PMR and existing application codes. Extensions to MPI and OpenMP standards will 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. This effort is tracked using the risk register.

Another risk is the failure of adoption of the software stack by the vendors, which is tracked in the risk register, and mitigated by the specific delivery focus in sub-element SW Ecosystems 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 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. This approach is being extended to other APIs and alternative models (that are being defined and eventually standardized) to allow for deeper and stack-wide introspection as well as resource sharing.

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. The duplication of effort specifically, and the overall integration of the software stack, are tracked in the risk register.

2.2 DEVELOPMENT TOOLS

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

2.2.1 Scope and Requirements

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

open APIs to provide portability, additional innovation, and integration into the tool suite and the overall software stack.

2.2.2 Assumptions and Feasibility

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

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

2.2.3 Objectives

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

2.2.4 Plan

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

2.2.5 Risks and Mitigations Strategies

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

Given its wide use within a range of different communities, and its modular design principles, the project's open source compiler activities will focus on the use of the LLVM compiler infrastructure as a path to reduce both scope and complexity risks and leverage with an already established path for NRE investments across multiple vendors. The compilers and their effectiveness are tracked in the risk register.

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

2.3 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.

2.3.1 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.

2.3.2 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, Mesquite, MOAB, 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.

2.3.3 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.

2.3.4 Plan

As detailed below, the Mathematical Libraries effort supports complementary projects as needed to meet the needs of ECP applications. To ensure relevance to DOE missions, these efforts are DOE-laboratory-led, including strong collaborations with academia, industry, and other organizations, and leveraging existing libraries that are widely used by the DOE HPC community.

Initial efforts focus 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 will also inform research into advanced exascale-specific numerical algorithms that will be implemented within the libraries and frameworks. The projects will implement their software on the CORAL, NERSC and ACES systems, and ultimately on initial Exascale systems, so that functionality, performance, and robustness can be evaluated by the applications teams and other elements of the software stack. Throughout the effort the applications teams and other elements of the software stack will evaluate and provide feedback on their functionality, performance, and robustness. These goals will be evaluated at least yearly based on milestones as well as joint milestone activities shared across the associated software stack activities by Application Development and Hardware and Integration project focus areas.

2.3.5 Risks and Mitigations Strategies

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

- Efficient implementation of new or refactored algorithms to meet Exascale computing requirements may introduce unanticipated requirements on programming environments. To mitigate this risk, effective communication is needed between projects in the Mathematical Libraries effort and projects tasked

with developing the programming environments. From the application perspective, this is specifically tracked in a specific AD risk the risk register. Additionally, the risks of an inadequate programming environment overall are tracked as a specific ST risk in the risk register.

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

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

2.4 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.

2.4.1 Scope and Requirements

Changes in the hardware architecture of Exascale supercomputers will render current approaches to data management, analysis and visualization obsolete, resulting in disruptive changes to the scientific workflow and rendering traditional checkpoint/restart methods infeasible. A major concern is that Exascale system concurrency is expected to grow by five or six orders of magnitude, yet system memory and input/output (I/O) bandwidth/persistent capacity are only expected to grow by one and two orders of magnitude, respectively. The reduced memory footprint per FLOP further complicates these problems, as does the move to a hierarchical memory structure. Scientific workflow currently depends on exporting simulation data off the supercomputer to persistent storage for post-hoc analysis.

On Exascale systems, the power cost of data movement and the worsening I/O bottleneck will make it necessary for most simulation data to be analyzed in situ, or on the supercomputer while the simulation is running. Furthermore, to meet power consumption and data bandwidth constraints, it will be necessary to sharply reduce the volume of data moved on the machine and especially the data that are exported to persistent storage. The combination of sharp data reduction and new analysis approaches heighten the importance of capturing data provenance (i.e., the record of what has been done to data) to support validation of results and post-hoc data analysis and visualization. Data and Visualization is the title for Data Management (DM) & Data Analytics and Visualization (DAV) activities in the Exascale project.

Data management (DM) activities address the severe I/O bottleneck and challenges of data movement by providing and improving storage system software; workflow support including provenance capture; and methods of data collection, reduction, organization and discovery.

Data analytics and visualization (DAV) are capabilities that enable scientific knowledge discovery. Data analytics refers to the process of transforming data into an information-rich form via mathematical or computational algorithms to promote better understanding. Visualization refers to the process of transforming scientific simulation and experimental data into images to facilitate visual understanding. Data analytics and visualization have broad scope as an integral part of scientific simulations and experiments; they are also a distinct separate service for scientific discovery, presentation and documentation purposes, as well as other uses like code debugging, performance analysis, and optimization.

The scope of activities falls into the following categories:

- Scalable storage software infrastructure – system software responsible for reliable storage and retrieval of data supporting checkpointing, data generation, and data analysis I/O workloads

- Workflow and provenance infrastructure – facilitating execution of complex computational science processes and the capture and management of information necessary to interpret and reproduce results
- Data collection, reduction, and transformation – enabling complex transformation and analysis of scientific data where it resides in the system and as part of data movement, in order to reduce the cost to solution
- Data organization and discovery – indexing and reorganizing data so that relevant items can be identified in a time- and power-efficient manner, and complex scientific data analysis can be performed efficiently on Exascale datasets
- In situ algorithms and infrastructure – performing DAV while data is still resident in memory as the simulation runs enabling automatic identification, selection and data reduction for Exascale applications.
- Interactive post-hoc approaches – on data extracts that produced in situ and support post-hoc understanding through exploration.
- Distributed memory multi-core and many-core approaches, for the portable, performant DM and DAV at Exascale.

2.4.2 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).

2.4.3 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.

2.4.4 Plan

Particularly in the area of DM, productization of technologies is a necessary step for adoption, research-quality software is not enough. One approach we will take is to fund vendors of products in related areas to integrate specific technologies into their product line. When developing objectives for this activity, a focus was placed on the availability of products that deliver these technologies on platforms of interest. Activities can be separated into two categories:

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

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

Portions of the DAV software stack are being productized and supported by industry, which will help to control costs in the long term. Activities to achieve the DAV objectives are heavily dependent on developments across the Exascale project, and thus close coordination with other teams is essential. Close engagement with application scientists is crucial to the success of DAV, both in terms of understanding and addressing the requirements of science at scale and ensuring that computational scientists are able to adopt and benefit from the DAV deliverables.

Many objectives need initial research projects to define plausible solutions. These solutions will be evaluated and progressively winnowed to select the best approaches for the Exascale machine and the needs of science. Selected projects will continue to receive support to extend their research and development efforts to integrate their solutions into the open-source Exascale software stack.

2.4.5 Risks and Mitigations Strategies

- 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 the risk register. The software stack tracks several risks indirectly related to data management in the risk register 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. This is tracked in the risk register.
- 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.
- The skilled workforce required for a successful Exascale project is tracked in in the risk register.

2.5 SW ECOSYSTEM & DELIVERY

End State: A production-ready software stack delivered to our facilities, vendor partners, and the open source HPC community.

2.5.1 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:

- 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) for use by ECP teams at the Facilities
- Hardening and broad ST and facility adoption of Spack for easy build of software on all target platforms
- Development and hardening of new methods for software deployment through the use of container technology
- Informal partnerships with the Linux Foundation's OpenHPC project for potential broader deployment of ST technologies in the OpenHPC ecosystem
- Co-design of ST solutions with (primarily ATDM) application teams, particularly in the area of programming models, abstractions for performance portability, and optimized use on Exascale systems
- System software that is typically provided by the vendors on a platform, or tightly integrated with vendor solutions – including resource managers, low-level runtimes, power management, and support for hierarchical memory at the Operating System (OS) level

- Development of Flang through a subcontract with NVIDIA – a first-of-its-kind open source Fortran compiler built on the LLVM toolchain
- Research in resilience to understand the impacts of faults on applications, software, and systems

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 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 sustainment required by applications, software, Facilities, and vendors who rely upon these products.

All three of the ATDM efforts in this area have a focus on integration of technologies into their new ATDM applications being developed "from scratch" under AD National Security Applications. ATDM applications are a high-risk, high-reward effort under tight timelines for delivery to the NNSA mission, and thus must focus on integration with key ST technologies being developed at those labs from the beginning.

System software in the form of operating systems capabilities and low-level runtimes has historically been built upon a node-centric viewpoint with a global view of the system tied together in a patchwork of add-on tools and resource managers. In order to support higher-level software development, low-level software layers must be provided to address hierarchical and non-uniform memory management, dynamic power management, lightweight threading and process management, low-level distributed data movement (i.e., messaging), I/O forwarding, and resilience and integrity issues. In addition, support for sophisticated resource scheduling and management, including storage, must account for the ability to accomplish increasingly complex workflows (e.g., ensembles, multi-physics, scale bridging, and uncertainly quantification).

The overarching goal of the resilience and integrity (RI) effort is to keep the application workload running to an acceptably correct solution in a timely and efficient manner on future systems, even in the presence of increasing failures, challenges in I/O scalability for checkpoint/restart, and silent (undetected) errors.

2.5.2 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. Recent restructuring of the ECP to formalize 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 system software stack as a feasible solution. We do however recognize that in many cases the features of today's HPC operating system 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 where user-level (i.e., non-kernel) solutions are developed.

2.5.3 Objectives

This area will focus on all aspects of integration of the ECP software stack, 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 continuous integration and containers. Likewise, we will aim to influence the deployment of operating system, low-level runtimes, and perhaps containers typically deployed by our vendor partners. Finally, our ATDM projects will focus on delivery and integration of novel software technologies into next-generation applications under development in AD National Security Applications – with a goal of demonstrating and hardening those technologies for possible use in other applications.

Additional goals include providing infrastructure and higher-level tools that address the requirements and extensions for better resource allocation and job scheduling capabilities. These changes will address the necessary aspects of system architectures, including storage resources, and the support for increasingly complex workflows that are projected to occur within the Exascale environment.

The Flang effort is being developed by NVIDIA's PGI compiler team based on the robust and widely used PGF compiler. Flang was released on GitHub as an open source project in 2017, and is making solid progress toward performance and portability goals. Our objective is to have Flang supported at the DOE Facilities for use by ECP application teams, as well as taken up by vendors (Arm being an early adopter) as a first-class Fortran solution

The objective of resilience efforts is that applications will run successfully and efficiently to timely completion in the presence of any faults experienced on the system. Application developers will have the necessary programming tools, libraries, and system support for incorporating resilience into their code. This will include access to nonvolatile memory, fault tolerant libraries, and scientific libraries that are resilient to soft errors and support application developers to implement their own resilient algorithms.

2.5.4 Plan

Initial efforts will be aimed at developing and deploying a continuous integration system at each of the DOE Facilities based on the results of a working group established in 2017 that defined the key gaps missing in current prototype CI solutions. In parallel with that effort, we will work closely with the five SDK projects in developing a long-term plan for software deployment. We will also begin efforts to evaluate existing container technologies (e.g., Bee, Argo, Singularity, others) built upon the widely-adopted Docker technology base, and build on facility collaborations established for CI deployment to develop one or more solutions for container deployment that meet the unique HPC requirements for performance and security. As Flang becomes competitive in features and performance with other Fortran compilers, we will work with AD projects on defining the scope of work for NVIDIA/PGI, and with the Facilities on deployment of recent Flang releases for broad community testing and use.

Our plan is to hire an ECP Release Engineer who can work closely with the individual SDK projects in each L3 area to develop common practices and delivery mechanisms. This could include a pathway for SDKs into OpenHPC, as well as well-publicized incremental releases of SDKs.

2.5.5 Risks and Mitigations Strategies

- Vendors unwilling to adopt aspects of the Argo environment that require kernel-level support.
- Inability to staff the release engineer position with a qualified candidate (someone with DevOps expertise who also understands the HPC environment).
- 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. Singularity isn't funded by ECP. BEE and Argo are the only ECP funded items, and it is unclear if they are the right final solution.
- Resilience work is a tiny fraction of effort. May need to consider a software co-design center around this topic that would marry efforts between apps, ST, and vendors.
- ATDM efforts may continue to be inward-facing and not suitable for longer-term broad adoption if/when technologies are successfully borne out in practice.
- OpenHPC partnership is ill-defined, and unfunded.
- Sustainability of ECP ST capabilities after ECP has ended.

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.

Spack Package Support	
Have Spack Package	43
Spack Package in progress	21

Source Build System	
Cmake	44
Configure/Make (autotools)	32
Custom	4

Delivery	
Direct to users from source	81
Vendor stack	11
ALCF	19
OLCF	20
NERSC	20
LLNL	18
LANL	17
OpenHPC	9
Containers (Docker)	3+

User Support	
Documentation	81
Tutorials	50
Support staff training	21
Email/phone contact	70
User-access issue tracking	65

Figure 9: The 54 ECP ST Projects contribute to 89 unique products. ECP ST products are delivered to users via many mechanisms. Provides experience we can leverage across projects. Building via Spack is required for participating in ECP ST releases: 48% of products already support Spack. 24% have Spack support in progress. Use of Spack and the ECP ST SDKs will greatly improve builds from source. 81 of 89 packages support users via source builds.

3.1 ECP ST PRODUCTS

ECP ST efforts contribute to 89 software products in five technical areas (Table 1). 33 of the 89 products are broadly used in the HPC community and require substantial investment and transformation in preparation for Exascale architectures. An additional 23 are important to some existing applications and typically represent new capabilities that enable new usage models for realizing the potential that Exascale platforms promise. The remaining products are in early development phases, addressing emerging challenges and opportunities that Exascale platforms present.

Product	Website	Deployment Scope
GASNet-EX	http://gasnet.lbl.gov	Broad
Kokkos	https://github.com/kokkos	Broad
MPICH	http://www.mpich.org	Broad
OpenMPI	https://www.open-mpi.org	Broad
RAJA	https://github.com/LLNL/RAJA	Broad
ROSE	https://github.com/rose-compiler	Broad
CHAI	https://github.com/LLNL/CHAI	Moderate
Global Arrays	http://hpc.pnl.gov/globalarrays	Moderate
Legion	http://legion.stanford.edu	Moderate
LLVM OpenMP compiler	https://github.com/SOLLVE	Moderate
OpenMP V & V Suite	https://bitbucket.org/crpl_cisc/sollve_vv/src	Moderate
Qthreads	https://github.com/Qthreads	Moderate
Umpire	https://github.com/LLNL/Umpire	Moderate
UPC++	http://upcxx.lbl.gov	Moderate
BOLT	https://github.com/pmodels/argobots	Experimental
DARMA	https://github.com/darma-tasking	Experimental
Intel GEOPM	https://geopm.github.io	Experimental
PaRSEC	http://icl.utk.edu/parsec	Experimental

Table 4: Programming Models and Runtimes Products (18 total).

Product	Website	Deployment Scope
Caliper	https://github.com/llnl/caliper	Broad
Dyninst Binary Tools Suite	http://www.paradyn.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
Tau	http://www.cs.uoregon.edu/research/tau	Broad
mpiFileUtils	https://github.com/hpc/mpifileutils	Moderate
openarc	https://ft.ornl.gov/research/openarc	Moderate
Papyrus	https://ft.ornl.gov/research/papyrus	Moderate
Program DB Toolkit (PDT)	https://www.cs.uoregon.edu/research/pdt	Moderate
TriBITS	https://tribits.org	Moderate
CHiLL Compiler		Experimental
Exascale Code Gen Toolkit		Experimental
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 5: Development Tools Products (19 total).

Product	Website	Deployment Scope
hypre	http://www.llnl.gov/casc/hypre	Broad
Kokkoskernels	https://github.com/kokkos/kokkos-kernels	Broad
MFEM	http://mfem.org/	Broad
PETSc/TAO	http://www.mcs.anl.gov/petsc	Broad
SLATE	http://icl.utk.edu/slate	Broad
SUNDIALS	https://computation.llnl.gov/projects/sundials	Broad
SuperLU	http://crd-legacy.lbl.gov/~xiaoye/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
ForTrilinos	https://trilinos.github.io/ForTrilinos	Experimental
libEnsemble	https://github.com/Libensemble/libensemble	Experimental
Tasmanian	http://tasmanian.ornl.gov	Experimental

Table 6: Mathematical Libraries Products (16 total).

Product	Website	Deployment Scope
Catalyst	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	https://www.paraview.org	Broad
ROMIO	http://www.mcs.anl.gov/projects/romio	Broad
VeloC	https://xgitlab.cels.anl.gov/ecp-veloc	Broad
VisIt	https://wci.llnl.gov/simulation/computer-codes/visit	Broad
VTK-m	http://m.vtk.org	Broad
ADIOS	https://github.com/ornladios/ADIOS2	Moderate
ASCENT	https://github.com/Alpine-DAV/ascent	Moderate
Cinema	https://datascience.lanl.gov/Cinema.html	Moderate
zfp	https://github.com/LLNL/zfp	Moderate
C2C		Experimental
FAODEL	https://github.com/faodel/faodel	Experimental
GUFI	https://github.com/mar-file-system/GUFI	Experimental
HXHIM	http://github.com/hpc/hxhim.git	Experimental
MarFS	https://github.com/mar-file-system/marfs	Experimental
Mercury	http://www.mcs.anl.gov/research/projects/mochi	Experimental
ROVER		Experimental
Siboka		Experimental
SZ	https://github.com/disheng222/SZ	Experimental
TuckerMPI		Experimental
UnifyCR	https://github.com/LLNL/UnifyCR	Experimental

Table 7: Visualization and Data Products (25 total).

Product	Website	Deployment Scope
Flang/LLVM	Fortran compiler http://www.flang-compiler.org	Broad
Spack	https://github.com/spack/spack	Broad
ArgoContainers	https://xgitlab.cels.anl.gov/argo/containers	Moderate
BEE		Experimental
FSEFI		Experimental
Sonar		Experimental
Secure JupyterHub		Experimental
Kitten Lightweight Kernel	https://github.com/HobbesOSR/kitten	Experimental
AML	https://xgitlab.cels.anl.gov/argo/aml	Experimental
COOLR	https://github.com/coolr-hpc	Experimental
NRM	https://xgitlab.cels.anl.gov/argo/nrm	Experimental

Table 8: Software Delivery and Ecosystems Products (11 total).

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 10 summarized ECP ST staff involvement in the major standards efforts that impact ECP.

ECP ST staff are heavily involved in MPI and OpenMP standards efforts. ECP ST staff hold several key leadership positions and have heavy involvement in all aspects. ECP ST staff also play a critical role 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 4.5.8, 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.

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 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 9 expressly eliminate this large cost for DOE because the product is developed and supported outside of DOE.

Product	Contribution
MAGMA	ECP ST math libraries efforts inform the design, implementation, and optimization of numerical linear algebra routines on NVIDIA GPUs
Compilers and runtime	The Validation and Verification Suite (on-going effort) for the SOLLVE project has helped uncover bugs in OpenMP implementations provided by Cray, LLVM and XL.
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.
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. An ECP ST staff member belongs to several working group related to scalability and resilience where, in addition to the discussions, implements proof-of-concept features in OpenMPI.
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 it's deployment in an experimental release.
OpenHPC	An ECP ST staff member serves on the OpenHPC Technical Steering Committee as a Component Development representative.
LLVM	An ECP ST staff member is co-leading design discussions around the parallel IR and loop-optimization infrastructure.

Table 9: 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.

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 10: 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.

4. ECP ST PROJECT SUMMARIES

This section of the ECP ST Capabilities Assessment Report provides two-page summaries of each funded project. The text provides a project overview and summarizes the key challenges, solution strategy, recent progress and next steps for the project.

4.1 PROGRAMMING MODELS & RUNTIMES

This section present projects in Programming Models & Runtimes.

4.1.1 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 4.5.1.

4.1.2 LANL ATDM Programming Models and Runtimes

Overview The LANL ATDM PMR effort is focusing on the development and use of advanced programming models for Advanced Technology Development and Mitigation 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.

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).

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 as well as Fortran interoperability for Legion based applications.

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.

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

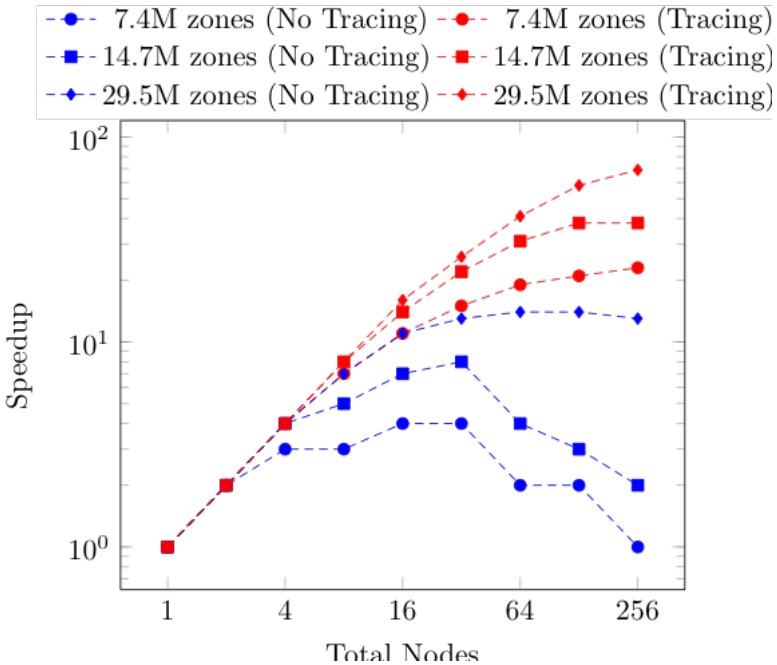


Figure 11: New Legion features such as dynamic tracing significantly improves strong scaling in unstructured mesh computations.

computation that is easy to reason about. However, the top-level task in this tree-of-tasks model can often become a sequential bottleneck, as it is responsible for the initial distribution of many subtasks across large machines. In earlier work NVIDIA developed the initial implementation of control replication, which allows the programmer to write tasks with sequential semantics that can be transparently replicated many times, as directed by the Legion mapper interface, and run in a scalable manner across many nodes. Dynamic control replication is an important capability for LANL's ATDM effort, allowing our application teams to write applications with apparently sequential semantics while enabling scalability to Exascale architectures. This approach will improve understandability of application code, productivity, and composability of software and ease the burden of optimization and porting to new architectures.

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 programing system in order to enable our Next Generation Code to be performance portable and scalable to future Exascale systems. Legion provides the underlying distributed and node-level runtime environment required for FleCSI to leverage task and data parallelism, data dependent execution, and runtime analysis of task dependencies to expose parallelism that would be tedious and error prone to expose at the application or middleware level. We completed an evaluation of the initial implementation of FleCSI on Legion using the FleCSALE hydrodynamics application.

Next Steps

FleCSI. Focus on performance and scalability enhancements of the Dynamic Control Replication and other new Legion features.

FleCSI Legion Integration. Demonstrate the integration of Dynamic Control Replication and other new Legion features within FleCSI. Our goal is to demonstrate a multi-scale application on the Advanced Technology System, Sierra using our latest advances in the Legion and FleCSI systems.

4.1.3 LLNL ATDM Programming Models and Runtimes

Overview This project covers two main thrusts in programming models standards and runtimes for exascale supercomputing systems. The first thrust is programming models standards work in MPI and OpenMP.

The Message Passing Interface Standard (MPI) is a message passing library standard based on the consensus of the MPI Forum, which has over 40 participating organizations, including vendors, researchers, software library developers, and users. The goal of the Message Passing Interface is to establish a portable, efficient, and flexible standard for message passing that will be widely used for writing message passing programs. As such, MPI is the first standardized, vendor independent, message passing library. The advantages of developing message passing software using MPI closely match the design goals of portability, efficiency, and flexibility. MPI is not an IEEE or ISO standard, but has in fact, become the "industry standard" for writing message passing programs on HPC platforms.

OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most HPC platforms. To enable portable tools for performance analysis and debugging of OpenMP programs, the OpenMP Language Committee is defining application programming interfaces for tools; these interfaces are expected to be part of the OpenMP standard and supported by all OpenMP compliant implementations. There are two parts to the proposed interfaces: OMPT (a first-party API for performance tools), and OMPD (a shared-library plugin for debuggers that enables a debugger to inspect and control execution of an OpenMP program).

The other main thrust area for LLNL is the ROSE project's work in support of ATDM Exascale application efforts. ROSE is an open source compiler infrastructure to build source-to-source program transformation and analysis tools for large-scale Fortran 77/95/2003, C, C++, OpenMP, and UPC applications. The intended users of ROSE could be either experienced compiler researchers or library and tool developers who may have minimal compiler experience. ROSE is particularly well suited for building custom tools for static analysis, program optimization, arbitrary program transformation, domain-specific optimizations, complex loop optimizations, performance analysis, and cyber-security.

Key Challenges

ROSE. We will develop advanced program transformation and analysis to improve correctness and performance of RAJA codes. The research results will be communicated to RAJA maintainers to improve the RAJA portable programming layer. Both of these LLNL efforts in this area are crucial for ECP and ASC applications to achieve portable performance on upcoming Exascale systems.

MPI. For MPI, we focus on interfaces for supporting tools, including MPI_T and MPIR, and for fault tolerance, including Reinit. Tools for MPI are critical to ensure that applications achieve high communication performance, and understanding and designing fault tolerance interfaces are important for large scale jobs on faulty systems. The participants will follow development across the standard in addition to the tools and fault tolerance areas.

OpenMP. In OpenMP, our focus is on the tools interfaces OMPT and OMPD. Tools are critical for application developers to understand the correctness and performance of their codes when using OpenMP. We will participate and monitor developments in all parts the standard in addition to our focus areas.

Solution Strategy

ROSE. The team is working to support advanced loop transformations on RAJA code; finding loops susceptible to data races in ASC applications; and performing performance analysis of RAJA code. ROSE improvements will be released as open source at https://github.com/rose-compiler/rose.

MPI. LLNL staff regularly participate in regular calls and discussions for the Tools Working Group, Fault Tolerance Working Group, Sessions Working Group and attend Forum meetings that occur during the quarter.

OpenMP. Regularly participate in weekly OpenMP language committee calls, participate in biweekly calls of the OpenMP tools WG, email discussions on OpenMP tools interfaces (OMPD and OMPT).

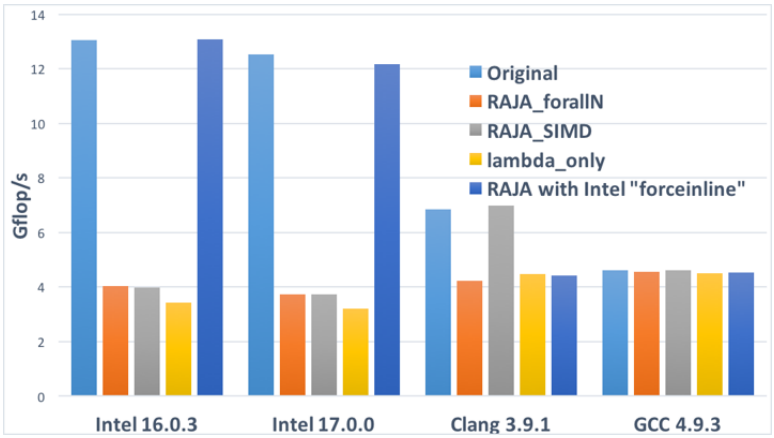


Figure 12: Work by ROSE team shows performance gap analysis for RAJA with different compilers.

Recent Progress

ROSE. The team continued to improve ROSE's C++11 support and the ROSE-based tool, rajaChecker. With bug fixes to ROSE, rajaChecker can process 1650 files in an ASC application without errors, compared to the original 1405 files. We also added new features into rajaChecker, such as recognizing user-defined RAJA wrappers, MIN/MAX macros, indirect loop index variables, etc. The latest test shows that rajaChecker finds 87 loops in an ASC application matching the data race pattern users requested, compared to the original 20.

MPI. The MPI participants participated in the design of new approaches to incorporate fault tolerance in the MPI Standard, including standardizing primitives to support fundamental fault-tolerance methods. They also worked on the MPI_T_Events interface for tools. The MPI participants attended the MPI Forum meeting in Portland in February and proposed new approaches to incorporate fault tolerance in the MPI Standard, including standardizing primitives to support fundamental fault-tolerance methods.

OpenMP. In OpenMP, the project members worked with the Tools subcommittee to prepare and enact several tickets to improve OMPT and OMPD specifications. The project members attended the OpenMP face-to-face meeting in Austin and worked with the Tools subcommittee to prepare and enact several tickets to improve OMPT and OMPD specifications.

Next Steps

ROSE. Continue to improve ROSE support for RAJA and ASC applications.
MPI. Continue to participate in MPI Forum on fault tolerance and tools.
OpenMP. Continue to participate in OpenMP Standards Committee on tools and debuggers.

4.1.4 SNL ATDM Programming Models: Kokkos

Overview Kokkos provides a C++ Parallel Programming Model for Performance Portability. It is implemented as a C++ abstraction layer over existing parallel programming models such as OpenMP, CUDA and C++ std::threads. Application and library developers can implement their code using Kokkos, which will map their parallel algorithms onto the underlying execution mechanism. Codes written in that manner can then be compiled to run on each current HPC platform, including CPU and GPU based systems. Furthermore, new types of architectures generally need only be added to Kokkos, while no changes are necessary to end applications - even if the new architecture requires its own custom programming mechanism.

Key Challenges Teams of scientists, engineers, and mathematicians apply their specific expertise to develop application software to solve problems from their specialized domains. The largest and most complex of these require the use of High Performance Computing (HPC). For more than two decades, HPC applications used parallel computing on a network of compute-nodes consisting of a computer chip with either a single core or very few processing cores and memory dedicated to each core. However, the "many-core revolution," which has slowly arisen in the last five years, has dramatically changed the nature of HPC because processors in modern compute-nodes have many cores that must share the compute-node's memory.

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.

A key issue for achieving performance portability is, that one does not only need to address parallel execution but also data management. This includes the handling of memory hierarchies consisting of resources such as HBM, DDR and NVM memory as well as the question of memory layouts, which might need to be different for various architectures.

Solution Strategy The Kokkos project 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.

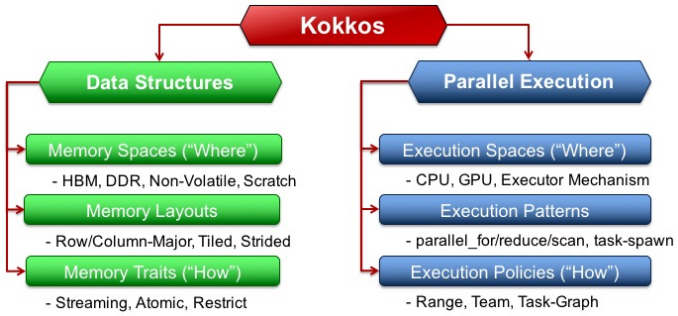


Figure 13: Kokkos Execution and Memory Abstractions

Applications written in Kokkos are leveraging its six core abstractions for parallel execution and data management. These abstractions provide the flexibility to allow the mapping of execution and data to the diverse set of architectures. As an example the "parallel_for" Execution Pattern does neither guarantee order of execution nor concurrency of execution. That allows Kokkos to use threading, vectorization or even pipelining of operations to parallelize the algorithm.

The Parallel Execution abstractions are (1) Execution Patterns, (2) Execution Policies, and (3) Execution Spaces. *Execution Patterns* describe what fundamental parallel algorithm is performed. This includes parallel loops, parallel reductions, and parallel scans as well as task spawn operations. *Execution Policies* control how the patterns are executed. They for example control the iteration space and whether scheduling is done dynamically or statically. *Execution Spaces* denote different resources to execute on such as GPU or CPU.

The Memory abstractions are (1) Memory Layout, (2) Memory Traits, and (3) Memory Space). *Memory Layout* describes the mapping of logical indices to the actual memory address. This allows to change the data access pattern in an algorithm without changing the implementation of the algorithm. *Memory Traits* describe access properties, such as whether the user wants to perform atomic accesses, or whether accesses are guaranteed non-aliased. *Memory Spaces* control where data is allocated. This allows users to access HBM and DDR memory as well as special Scratch Memory.

As a programming model, Kokkos is highly invasive in any code it is used by. Consequently, software quality is a critical concern. Kokkos addresses this through a comprehensive testing regime, which includes over 250 combinations of compilers, hardware platforms and backends which are run every night. These tests cover current production HPC environments as well as prototype environments for future systems.

Kokkos is available to users under the BSD license, permitting use of Kokkos in other open source codes as well as closed source code. It is maintained and developed at *https://github.com/kokkos/kokkos*.

Recent Progress Kokkos is now used by a wide number of applications at various institutions. This includes multiple DOE laboratories as well as universities and other HPC centers. At Sandia National Laboratories Kokkos is the primary programming model to address support for GPU based platforms in a performance portable manner, and numerous production level applications are actively porting to using Kokkos. The SNL ATDM applications are largely demonstrating performance portability today for a subset of their capabilities and are working towards full fletched support of all HPC platforms.

Recent advances in Kokkos include support for more portable and performant scatter-add type algorithms, dynamic and static task graph support as well as support for NIVDIA Volta and ARM CPU architectures. This enables Kokkos applications to run on the DOE Summit and Sierra machines, which are getting installed now as well as at the upcoming Vanguard ARM based machine to be installed at Sandia in Summer 2018.

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

4.1.5 SNL ATDM Programming Models: DARMA

Overview DARMA (Distributed, Asynchronous, Resilient Models for Applications) embeds safe and performant asynchronous tasking constructs in C++. DARMA defines asynchronous semantics for C++, providing a futures-based programming model. DARMA, however, provides additional semantics that carry extra safety and performance guarantees, including race freedom, fully non-blocking execution, and simplified load balancing and communication overlap. These semantics are implemented as a standards-compliant C++ header library, embedding asynchronous execution via metaprogramming.

Key Challenges Dynamic applications can be characterized by either unknown, imbalanced, or rapidly varying computation loads and communication patterns. Particle-in-cell, for example, has varying computational load per mesh region. As particles migrate, it also requires dynamic termination detection before entering a field solve phase. Overlap detection in contact algorithms rapidly creates load imbalance, which must be mitigated dynamically, in regions where contact occurs. Addressing these algorithmic challenges can involve:

- Latency-hiding of communication by trying to improve overlap with computation
- Elastic tasks with built-in data parallelism that can leverage more cores for larger tasks
- Asynchronous quiescence detection to avoid global synchronizations and minimize communication
- Semi-static load balancing (globally synchronous) for problems with persistent load imbalance
- Dynamic load balancing (work stealing) for problems with rapidly varying load imbalance

One possible option is for application developers to pursue domain-specific, ad hoc solutions. Managing things like resource contention between elastic tasks or implementing spanning trees for quiescence are highly non-trivial. A far more sustainable and productive approach would be providing these execution patterns in a reusable library with implementation complexities hidden behind a productive programming model. Programming should emphasize problem decomposition, data effect specification for correctness, and domain-specific cost models and heuristics for performance portability. The key challenges are both defining the programming abstractions to be used at application-level and implementing these dynamic execution constructs at the runtime-level.

Solution Strategy Similar to standard futures that create data-flow for C++ datatypes, DARMA provides asynchronous pointers with well-defined semantics that augment data-flow with non-blocking and race freedom guarantees. DARMA asynchronous pointers cab define serialization for arbitrary C++ types, enabling load balancing and asynchronous communication overlap. DARMA asynchronous pointers are intended to express concurrency in a sufficiently general way that multiple runtimes could serve as the backend implementation (Figure 14). To date, HPX (futures/promises), Charm++ (actor model), and MPI + OpenMP have all been able to provide relatively lightweight implementations despite their obvious design differences.

Another general approach consistent with the DARMA model for reducing algorithmic complexity is task *over-decomposition* in the context of shared-memory parallelism. For example, taking an MPI rank as the fundamental unit, over-decomposition would create many (e.g. 4-16) regions or patches per MPI rank, rather than a single large patch as would be most common. Overdecomposition aims to avoid complexities in explicit, application-level load balancing and explicit, application-level communication latency hiding. When load imbalance occurs, entire tasks can be migrated without requiring re-meshing or updating data structures. Similarly, communication occurs transparently through task dependencies, naturally pipelining and overlapping communication without requiring explicit Isend/wait constructs.

DARMA allows algorithms to be expressed with flexible granularity, tuning overdecomposition factors to meet either a hardware requirement (e.g. optimal balance of communication overlap and scheduling overhead, task blocking to fit caches) or an algorithmic requirement (e.g. sufficiently fine granularity for load balancing). Recent DARMA-Kokkos integration now allows over-decomposition on a per-node basis, rather than per-core. This greatly minimizes task scheduling overheads and can improve communication throughput (surface area-to-volume considerations). This further improves load balancing flexibility by allowing DARMA to control not only the number of tasks per node but the number of cores allocated to each task.

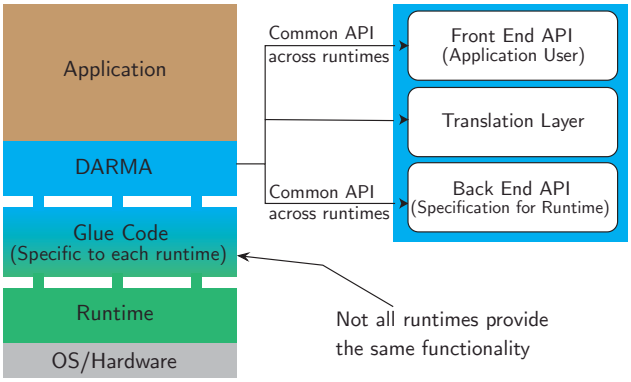


Figure 14: DARMA software stack model showing application-level code implemented with asynchronous programming model (DARMA header library). Application-level semantics are translated into a task graph specification via metaprogramming in the translation layer. Glue code maps task graph specification to individual runtime libraries. Current backend implementations include std::threads, Charm++, MPI + OpenMP, and HPX.

Recent Progress DARMA constructs are being used in the development of ATDM applications, including particle-in-cell (as part of EMPIRE), multiscale physics (in the ATDM tech demonstrator), and contact applications. MPI interoperability constructs have been defined and recently implemented. DARMA has integrated with Kokkos, providing support for data-parallel tasks.

Next Steps The next steps are pursuing both continued adoption in further applications, continued implementation of the programming model with additional backends, and continued improvement of the MPI + OpenMP reference implementation. For application adoption, a major focus will be using the MPI interoperability constructs to interface Trilinos solvers with DARMA kernels. Code improvements of the MPI reference implementation will focus on more load balancing heuristics, performance and correctness debugging, and demonstration at increasing scales on capability platforms like Trinity, Cori, and Sierra. DARMA is also active on the C++ standards committee, particularly for the parallelism technical specification. DARMA-inspired library features are being actively proposed and considered.

4.1.6 xGA

Overview The xGA project is focused on improving the performance, scalability, and user productivity of the Global Arrays (GA) library for exascale systems. This is essential to the ECP applications that already depend on GA such as NWChemEx[10], GAMESS[11] and GridPACK[12] (used as part of the Stochastic Grid Dynamics project). In addition, GA is being considered for use in the ECP application QMCPACK[13].

GA supports a shared memory-like programming model on distributed memory platforms. This allows users to create distributed multidimensional arrays that can be accessed from any processor using simple one-sided put/get/accumulate communication primitives. Data consistency is maintained using global synchronization mechanisms that flush all outstanding communication from the system and guarantee that arrays are in a known state. We are extending the GA library in a number of ways to take advantage of exascale architecture features including deep memory hierarchies and accelerators, while continuing to tune and improve the performance of the GA runtime.

Key Challenges Application codes increasingly need to take advantage of emerging hardware features to support more sophisticated scientific models. These hardware features extend the computing model beyond the concept of one compute thread per communication process as well as extending the memory model beyond the idea of a single flat view of data. Some of these Exascale platform features include massively multi-core CPUs, improved network communication speeds, the availability of GPU accelerators, and deep memory hierarchies. Any one or more of these capabilities promises to accelerate application codes, so long as these features are supported by the underlying runtimes and are accessible through easy-to-use interfaces.

Partitioned Global Address Space (PGAS) models have emerged as a popular alternative to MPI models for designing scalable applications. These are typically paired with a one-sided communication model and are particularly well-suited for applications with irregular access patterns that are not easily predicted in advance. Though MPI implementations are modernizing with features such as GPUDirect technology, PGAS models have not maintained similar advances. Users are requesting that PGAS models provide similar, seamless features for addressing and communicating with memory that may be located anywhere within the memory hierarchy, including memory resident on accelerators. They are also interested in using communication calls from multiple independent threads within a single traditional communication process. The lack of a consensus model for deep memory hierarchies makes developing a single interface for exploiting these by GA extremely difficult. Supporting multi-threaded applications in a robust and efficient manner is also likely to require significant software engineering in order to remove locks and achieve high performance.

Solution Strategy The xGA project has three primary thrusts:

1. **Thread safety:** GA has not previously provided any thread safety features or specification.
2. **Deep memory:** Users should provide hints to allocate GA memory anywhere within the memory hierarchy, including NVMe and GPUs.
3. **Application integration:** Any new feature should directly benefit one or more ECP applications.

Recent Progress

1. **Thread safety:** GA has not previously provided any thread safety features or specification. We have recently implemented a preliminary strategy for thread safety that puts locks on all communication calls. In principle, this could be fairly efficient, since communications calls are not all occurring at the same time, but the overhead of creating and destroying the locks themselves seems to be a severe drag on performance. We are currently reviewing individual calls to remove global variables where possible with a view to creating lock-free versions of these calls that can be used by a multi-threaded application.
2. **Deep memory:** We have begun looking at developing a strategy for creating GAs on GPU memory.
3. **Application integration:** GA is used in three applications currently supported by ECP. These include NWChemEx (via the TAMMS runtime layer), Stochastic Grid Optimization (via GridPACK) and GAMESS. We are also in discussions with QMCPACK application about using GA to support large

distributed lookup tables of spline coefficients. We expect that QMCPACK, as well as potentially NWChemEx, would benefit from a read-only property that we have been working on within GA. We are also discussing with NWChemEx how to potentially use thread-safe calls in GA as well as exploiting the deep memory hierarchy.

4. **Data layout:** We have developed new options for controlling the data layout within GAs. The internal routines for accessing both local and remote data have been completely reorganized to use an iterator model that has eliminated a lot of redundant code in the current implementation. It has also allowed us to implement a new 'tiled' array option guarantees that data is laid out in user-controlled blocks that are contiguous in memory. These layouts match layouts that are being considered by some of the linear algebra libraries and provide options for optimizing communication by eliminating strided communication calls. We have also begun a preliminary implementation of a sparse data layout designed to support sparse 2D array operations. We have used this to implement a sparse matrix-vector multiplication operation.
5. **Improved Performance:** We are continually evaluating and improving our performance. Figure 15 shows how we recently improved the bandwidth of our strided get operation in our 5.7 release of the Progress Rank ComEx runtime[14].

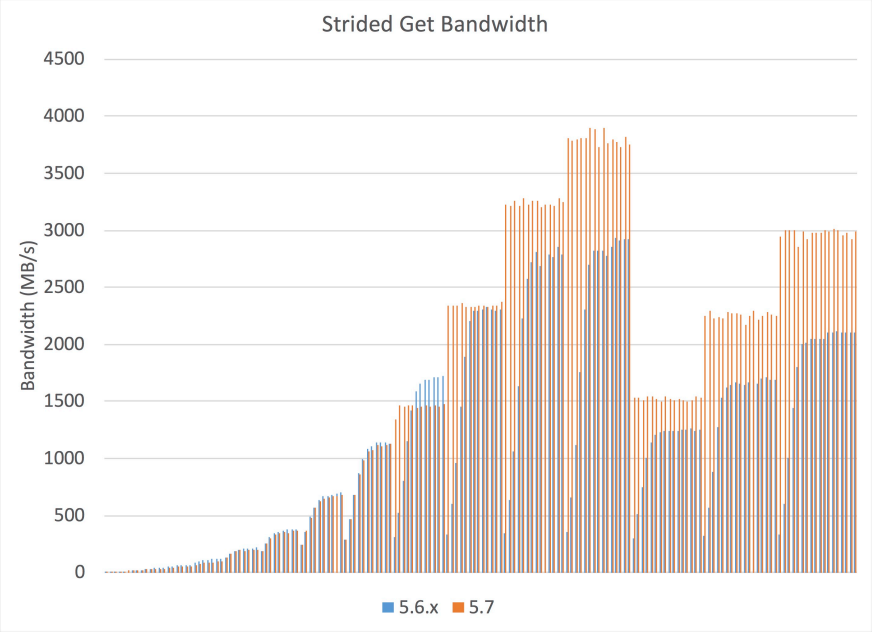


Figure 15: Improved performance of strided get in the 5.7 release series.

Next Steps Our next steps are:

1. **Add hints targeting memory hierarchy allocation:** xGA will expand its array property types allowing users to specify where memory should be allocated. We will also continue to refine the read-only property to include caching of requests so that we can expand the property to handle very large arrays. Currently, the property is limited to arrays that can be stored on a single node.
2. **Improved thread-safe performance:** We will improve our thread-safe performance by eliminating the use of globally shared variables and the locks that surround one-sided GA operations. In some cases, it may not be possible to implement thread-safe solutions without locks with the existing interface. In these cases, we will look to extending the interface to provide thread-safe alternatives that will deliver high performance.

4.1.7 ISC4MCM (RAJA)

Overview. The Integrated Software Components for Managing Computation and Memory Interplay at Exascale (ISC4MCM) project is 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 execution 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 for resource discovery, memory provisioning, allocation, access, operations, 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). 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), and SUNDIALS.

The software projects are highly-leveraged with other efforts. Team members include: ASC and ATDM application developers, ASD tool developers, university collaborators, and vendors. This ECP ST project supports outreach to the ECP community and collaboration with ECP efforts.

Key Challenges. The main technical challenge for this project is enabling production applications to achieve performance portability in an environment of rapidly changing, disruptive HPC hardware architecture design. Typical large applications contain $O(10^5) - O(10^6)$ lines of code and $O(10K)$ loop kernels. The codes must run efficiently on platforms ranging from laptops to commodity clusters to large HPC platforms. The codes are long-lived and are used daily for decades, so they must be portable across machine generations. Also, the codes are under continual development, with a steady stream of new capabilities added throughout their lifetimes – continual validation and verification is essential, which precludes substantial rewrites from scratch. Lastly, the complex interplay of multiple physics packages and dozens of libraries makes it so that the data required for the full set of components needed for a given simulation may not fit into a single system memory space. To advance scientific computing capabilities, applications must navigate these constraints while facing substantial hardware architecture disruption along the road toward Exascale computing platforms.

While the software provided by this project has a substantial user base at LLNL, achieving broader adoption in the ECP (projects without LLNL involvement, in particular) is another challenge. The software efforts are funded almost entirely by LLNL programs and the majority of their developers work on LLNL application projects. So resource limitations is a key issue.

Solution Strategy. The software libraries in this project focus on encapsulation and application-facing APIs to insulate users from the complexity and challenges associated with diverse forms of parallelism and heterogeneous memory systems. This approach allows users to exploit new capabilities with manageable rewriting of their applications.

RAJA provides various C++ abstractions for parallel loop execution. It supports: various parallel programming model back-ends, such as OpenMP (CPU multithreading and target offload), CUDA, Intel

Threading Building Blocks, etc.; loop iteration space and data view constructs to reorder, aggregate, tile, and partition loop iterations; complex loop kernel transformations for optimization, such as reordering loop nests, fusing loops, etc. RAJA also supports portable atomic operations, parallel scans, and CPU and GPU shared memory. After loops have been converted to RAJA, developers can explore implementation alternatives via RAJA features without altering loop kernels at the application level.

CHAI provides C++ "managed array" abstractions that automatically copy data to execution memory spaces as needed at run time based on RAJA execution contexts. Access to array data in loop kernels looks the same as when using traditional C-style arrays.

Umpire provides a portable API for managing complex memory resources by providing uniform access to other libraries and utilities that provide system-specific capabilities. Umpire decouples resource allocation from specific memory spaces, allocators, and operations. The memory introspection functionality of Umpire enables applications and libraries to make memory usage decisions based on allocation properties (size, location, sharing between packages, etc.)

All three software libraries are open source and available on GitHub [15, 16, 17]. There they provide regular software and documentation releases. Each project has dedicated email lists, issue tracking, test suites, and automated testing.

Recent Progress In FY18, CHAI and Umpire have been released as open source software projects and they are now developed on GitHub Recent development has focused on user documentation and cleaner integration of these two libraries to give applications more flexible and easy access to their capabilities.

Many new features have been added to RAJA in FY18 to enable flexible loop transformations for complex loop kernels via execution policies. LLNL applications are assessing this new functionality now in a "pre-release" version; it will be generally available before the end of FY18.

The RAJA Performance Suite [18] was released and made available on Github in January 2018. The Suite is used to assess and track performance of RAJA across programming models and diverse loop kernels. It is also being used for compiler acceptance testing in the CORAL procurement and was prepared for use as a benchmark for the CORAL-2 procurement.

In 2018, the RAJA project expanded its visibility beyond DOE NNSA Labs. Recent presentations include a RAJA tutorial at the 2018 ECP Annual Meeting and an application use case study the 2018 NVIDIA GPU Tech Conference (GTC). Future tutorials are planned at 2018 ATPESC and GTC 2019. Also, a RAJA paper and 1/2-day tutorial proposal were submitted to SC18.

Next Steps Our next efforts include:

1. **Fill RAJA Gaps:** Not all features are available for all programming model back-ends; as models mature, such as OpenMP4.5, these gaps will be filled.
2. **Expand RAJA User Guide and Tutorial:** Build example codes and user documentation for latest RAJA features and prepare for future tutorials (ATPESC 2018 and SC18).
3. **Expand RAJA Performance Suite:** Include kernels that exercise more application use cases and RAJA features.
4. **Focus RAJA Vendor Interaction:** Work with CORAL vendors to address issues as applications port to the Sierra platform at LLNL; establish early interactions with CORAL-2 vendors to ensure RAJA will be supported well on CORAL-2 systems.
5. **CHAI-Umpire Integration:** Enable applications to customize CHAI array allocations; e.g., to use memory pools for temporaries.
6. **Expand Umpire Capabilities:** Explore potential collaboration with relevant ECP efforts, such as SICM project.

4.1.8 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.

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 [19, 20, 21] (such as OpenMP, OpenACC and emerging asynchronous task models);
2. Scalability and performance over complex architectures [22, 23, 21, 24] (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 [25]. Apart from this, we also looked at communication strategies which allow the MPI library to be as lightweight as possible [26, 27]. Other ongoing 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 [28, 29, 30] started with the survey on heterogeneous memory architectures on upcoming DOE machines and how MPICH can take advantage of them [31]. The ongoing efforts include the investigation of utilizing heterogeneous memory inside the MPI implementation and evaluation of applications.

Exascale MPI topology awareness efforts [32, 33] originated with the investigation and evaluation of hints based on topology awareness and optimizations to virtual topology functionality in MPICH [34, 35]. The ongoing efforts include investigation of topology-aware collectives and neighborhood collectives in MPICH [36] and evaluation of the selected ECP applications.

Exascale MPI fault tolerance efforts [37, 21] started with support for handling noncatastrophic errors in MPI. The second effort included defining the scope of errors in MPI, a prerequisite for user-level failure mitigation (ULFM). Other efforts that we are looking at in this direction are standardizing ULFM in MPI and evaluating application suitability for fault tolerance.

Exascale MPI+X hybrid programming developed firstly with effort in improving interoperation of MPICH with threads [38]. Secondly, we included support for interaction of MPICH with user-level thread (ULT) libraries [39], primarily targeting Argobots and the BOLT runtime and the work-queue data transfer model for multithreaded MPI communication. Other issues that are being looked at include the investigation and evaluation on interaction between MPI and OpenMP and the study and evaluation of MPI endpoints.

Recent Progress Figure 16 provides the details of the milestones completed in FY18Q1 and FY18Q2. Milestones completed in December 2017 include: (1) classifying what noncatastrophic errors; when noncatastrophic errors occur (e.g., upon exhaustion of resources), our implementation informs the user of the error without disrupting MPI functionality. (2) topology-aware communicator split for NUMA, cache, and other on-node hardware resources, as well as for storage systems [34, 35], such that the newly created communicators are local to processes sharing hardware resources. (3) error scope definition as a prerequisite for user-level failure mitigation; we clarified how errors should be handled when they are not attached to a communicator, window, or file.

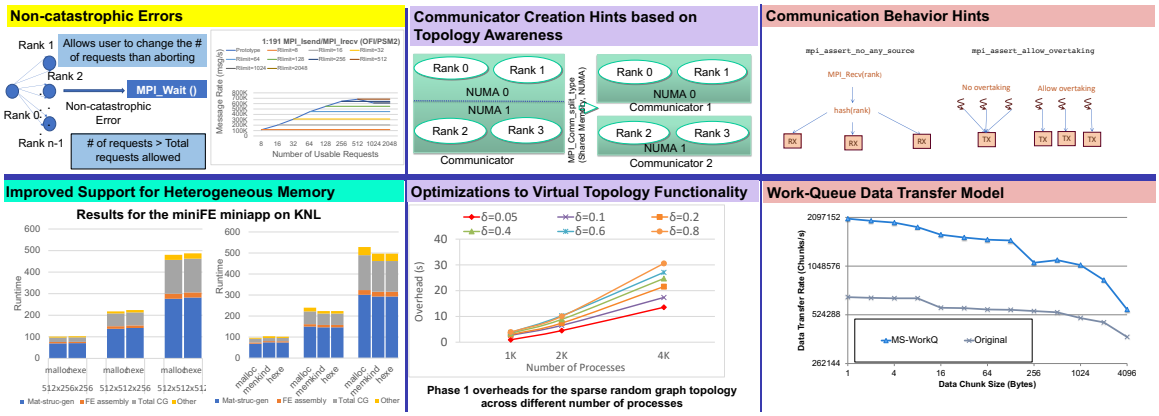


Figure 16: The details of MPICH milestones completed in FY18Q1 and FY18Q2

Milestones completed in March 2018 include: (1) investigating the memory subsystem of future ECP machines to understand the possible memory architectures; we presented and evaluated Hexe, our new framework that improves both the flexibility and portability of memory allocation and management of heterogeneous memory compared with other memory allocation tools such as malloc and memkind [31]. (2) designing the interface and algorithm for remapping processes and nodes according to the topology or allocating resources based on application communication patterns, in terms of MPI graph and Cartesian functions. (3) work-queues to hold work descriptors of work-generating threads; this approach decouples work-generating threads from completion-waiting ones and avoids interference between them.

Next Steps Several follow-on efforts are planned for the project:

1. **Performance and Scalability:** Exascale MPI efforts include investigation and evaluation of communication relaxation hints, investigation of optimizations to memory scalability in MPICH and improvements to MPI RMA operations.

2. **Heterogeneity:** Exascale MPI efforts include the investigation of utilizing heterogeneous memory inside the MPI implementation and ECP application performance evaluation of utilizing heterogeneous memory inside the MPI implementation.

3. **Topology Awareness:** Exascale MPI efforts include the analysis of the performance evaluation of the implementation of virtual topology functionality, investigating topology-aware collectives and neighborhood collectives in MPICH and evaluate the selected ECP applications.

4. **Fault Tolerance:** Exascale MPI effort is to investigate for fault tolerance with the focus on evaluating applications suitability for ULFM in MPI, to focus on the standardization of ULFM in MPI and finally, investigate and evaluate the implementation of MPI-4 ULFM proposal in MPICH.

5. **MPI+X Hybrid Programming:** Exascale MPI efforts include the investigation and evaluation on interaction between MPI and OpenMP, to investigate a study of the benefit potential of the MPI endpoints approach and evaluate it with selected ECP applications.

4.1.9 Legion

Overview The Legion project focuses on the development of the Legion runtime system and programming model. Our focus is on providing the base capabilities of an alternative task-based programming model that seeks to improve the amount of available parallelism and enable a separation of concerns of the implementation of a task from how that task and its associated data are mapped onto a given system architecture. Our efforts have focused on addressing bugs and adding new features to the implementation but also supporting applications that are interested in or already using Legion.

The Legion programming system is freely available via an open source license on the GitLab site: https://gitlab.com/StanfordLegion/legion.

Key Challenges Legion focuses on providing a programming model and supporting implementation that will help address the challenges applications will face in realizing sustained performance on what are projected to be the nature of Exascale systems. Increased scales combined with the challenges of programming potentially diverse accelerator and node-level processor technologies are responsible for a significant challenge that has yet to be fully addressed by today's most prominent programming systems – none of which have been fully validated on *yet-to-be-determined* system architectures for the Exascale era of computing.

Solution Strategy In funded collaboration between Los Alamos and Stanford University we are providing not only the implementation of the Legion programming model but also numerous opportunities for application developers and participants in the ECP PathForward efforts to learn about Legion and data-flow and task-based approaches to programming. We also closely work with Combustion-Pele (AD 2.2.2.02) and the data analytics efforts in support of ExaFEL (AD 2.2.4.05) to provide bug fixes, performance optimizations and implementation help. We work with these applications and are exploring Legion with a few others, and in collaboration with the LANL ATDM Programming Models and Runtimes project (ST 2.3.1.02), to identify needs and missing features in the programming model and runtime implementation. In addition, we also look at numerous aspects of having the Legion system interoperate with today's more widely used programming systems – e.g. MPI and OpenMP. This is critical in terms of providing a path for adoption and experimentation to occur in a more productive fashion. Finally, we are also actively exploring techniques for simplifying Legion programming to help assist in not only potential adoption but also in helping to educate the broader community about the programming model and its advantages on Exascale-class systems.

Recent Progress Our most recent progress has been devoted to getting the S3D DNS combustion application running on the Summit system at OLCF and the Piz Daint system at the Swiss National Supercomputing Centre. This work is utilizing the most recent version of Legion with a goal of both bug, scaling and overall performance enhancements. This effort is an update that covers new science relative to our previous work that has recently been published [40]. At present the full set of performance and application characteristics for this effort are still being analyzed and in particular, issues encountered on Summit are being discussed with OLCF staff. A early look at rough numbers suggest anywhere from a 26 to over 100X boost in performance over the *production* (MPI-based) version of S3D. Additional Legion features worked on in collaboration with LANL's ATDM efforts (ST 2.3.1.02) have resulted in much improved scaling capabilities due to reduced runtime overheads in comparison to past work.

Figure 17 on the following page, shows the task graph for a single time step on one node of the Legion implementation of S3D simulating an n-dodecane reaction. We hope to soon be able to release a more detailed analysis of the new Legion-S3D runs.

Next Steps We will continue to focus on improving the interoperability of Legion with other programming systems, simplifying the programming API for the Legion runtime to improve both our educational outreach as well as developer productivity, have regular open source releases of Legion and also work with application teams for debugging, feature improvements and performance tuning. In addition we are actively monitoring the emerging hardware technology components that are potential targets for use in the Exascale systems that will be eventually deployed by both the DOE Office of Science and the NNSA.

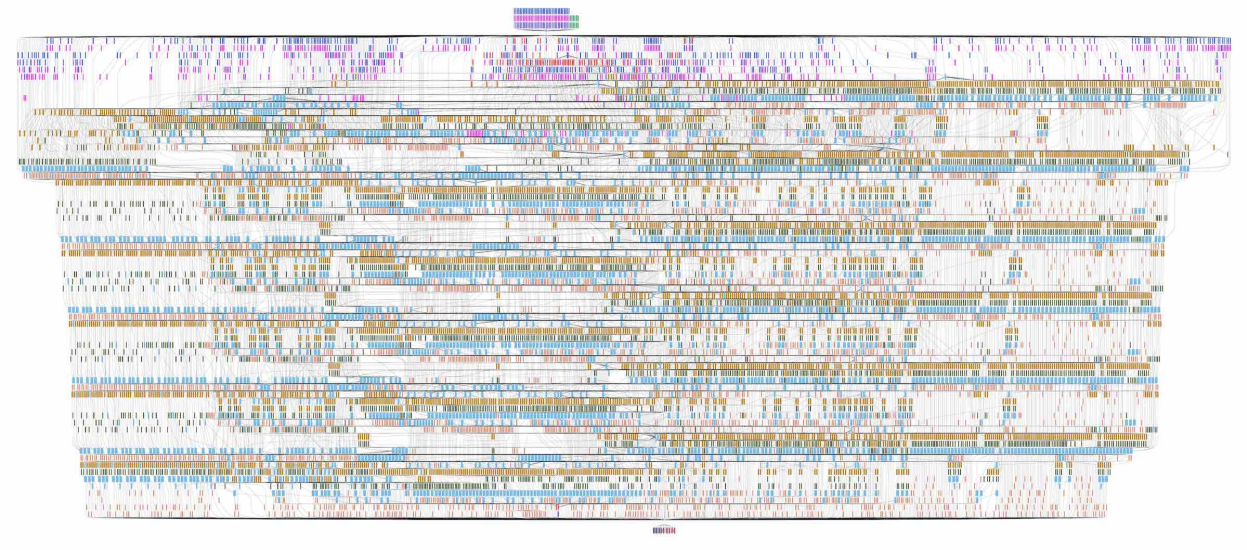


Figure 17: The Legion task graph for a single time step on a single node. The S3D configuration in this example is simulating n-dodecane chemistry reactions in addition to the direct numerical simulation of the turbulent flow.

4.1.10 Distributed Tasking at Exascale: PaRSEC

Overview The PaRSEC Environment provides a runtime component to dynamically execute 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 we create an execution model that enables developers to express as much parallelism as possible in their applications, so that those applications effectively utilize the massive collection of heterogeneous devices that ECP machines will deploy. Second, how we ensure that that execution model is flexible and portable enough to actually increase the scientific productivity of those same application developers, not only for the ECP target environments but for the foreseeable future. PaRSEC is an open source, community-based implementation of a generic task-based runtime that is freely available, and used by an increasing number of software libraries. The PARSEC development team is mainly comprised of research staff at UTK, but regular contributions from the community are provided via our presence on GitHub and Bitbucket The project focuses on prototyping different approaches to define task-based languages that will be 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 boundaries, or with particular programming languages. The DTE project provides means to maintain a high competitiveness in the field leading to more innovation on addressing the challenges we are facing toward scalable, performant and Exascale ready programming paradigms.

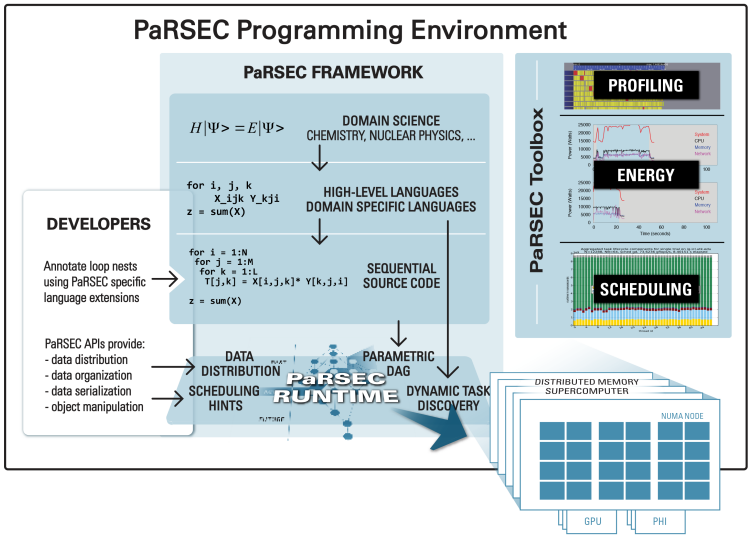


Figure 18: PaRSEC architecture

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

Solution Strategy The approach followed in PaRSEC is to provide a low-level, flexible and dynamic runtime able not only to schedule tasks at the node level, but to handle data transfers between different memory (both inter and intra nodes), memory hierarchies, heterogeneous architectures with support for accelerators with a simple programming scheme. The proposed approach envisions a middle-ground solution, addressing both hardware and software challenges. At the hardware level a team of dedicated developers extends PaRSEC to map it's capabilities to the hardware and to improve it's scalability and performance. At the upper level the interaction with the users is through building 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.

Recent Progress The runtime system of PaRSEC has been extended to provide a better support of dynamic task pools (DAGs of tasks), opening the runtime to new classes of algorithms, such as those where

the total number of tasks is a priori unknown. We enabled such dynamic DAGs by removing internal constraints on how tasks are identified and how dependencies are tracked. However, one of the missing capabilities, the detection of the application termination, needed to be provided explicitly. We added a set of termination detection algorithms to remove this step with better efficiency, and providing a complete solution to handling dynamic DAGs.

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

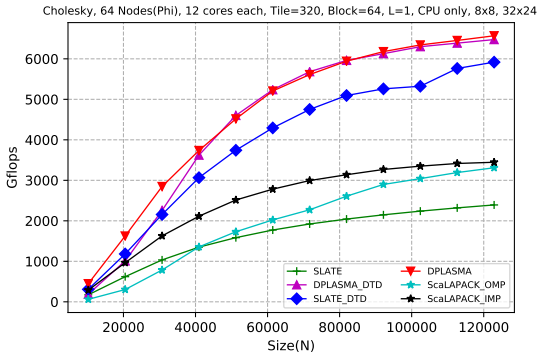


Figure 19: PaRSEC architecture

In figure 19, we compare the integration of SLATE and PaRSEC against the state of the art. First against the two legacy domain specific languages that have the capability to do linear algebra; then against the regular SLATE using OpenMP for intra-node parallelism, and MPI for communication; and finally against ScaLAPACK, which is the reference for distributed linear algebra.

On the software quality side, the PaRSEC runtime has been evaluated and amended to compile and run on all major target pre-Exascale platforms (ALCF Mira, Theta; OLCF Titan, Summit-dev). In particular the detection of architecture specific features on NVIDIA V100 accelerators and Power processors has been improved. PaRSEC now includes a Spack definition file to ease the deployment on future target systems as part of the system software SDK effort.

Next Steps A major effort at refactoring the accelerator and GPU component of the PaRSEC runtime is ongoing. The new runtime component handles memory transfers to/from accelerators as schedulable completion events that can trigger asynchronously the next step of the accelerated task and/or dependent tasks. The new engine is modular, one of the component provides a CUDA backend optimized for NVIDIA accelerators, and a new component will provide an OpenMP target backend for portability to other hardware.

We are improving the performance characterization system of PaRSEC to enable users to get detailed information on the performance of their applications that use PaRSEC as a runtime system. Part of this effort is shared with the development of Software Defined Events in ExaPAPI and the integration within the PaRSEC runtime.

The PaRSEC runtime will be augmented with the capability for any task to decide that a given algorithm has no more reason to be executed. This feature requires the runtime to properly remove any trace of the algorithm without going into extreme solutions. We do not want to dry run the algorithm without actually executing anything, and we do not want to remove the task from the runtime without any consideration for the other nodes. The solution will have to be a local decision that will leave the runtime in a coherent state without impeding the performance with the execution of useless tasks.

The detection of collective pattern at the runtime level is a hard problem. To enhance the insert task interface, the PaRSEC development team will provide a communication API. It will give the developer the capability to express communication where and when it seems fit during the algorithm.

4.1.11 Kokkos Support

Overview Kokkos Support provides documentation, training and community building support for the Kokkos programming model. To that end, the project develops programming guides, API references and tutorial material for Kokkos which is presented both as independent events and at conferences such as Supercomputing. Kokkos Support is also responsible for setting up community interaction channels such as the Slack channels now used for user communication and fostering the GitHub interactions between Kokkos developers and users. Finally, the project supports engagement with the C++ standard in order to further the adoption of successful Kokkos concepts into the core language.

Key Challenges For a new programming model to be successful, a comprehensive support and training infrastructure is absolutely critical. 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 Kokkos Support addresses these challenges through a number of ways. 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 and Los Alamos laboratories which can serve as direct contacts for local applications and, in Oak Ridge's case, for users of the Oak Ridge Leadership Computing Facility. Secondly, the project develops 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 an 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.

Recent Progress Kokkos Support has successfully run multiple Kokkos bootcamps for DOE applications as well as organized a number of single day tutorials. At the most recent tutorial during the DOE ECP All Hands meeting, the new cloud-based hands-on infrastructure was used for the first time, allowing tutorial attendees to use GPUs on remote servers without the hassle of temporary user account administration at DOE computing facilities. The project also improved existing documentation and transferred it to GitHub wiki pages which are tailored for software documentation and more maintainable. The Slack channels usage is growing, which have seen participation from users across the DOE and Kokkos community. There are also more interactions on GitHub issues, including a number of pull requests volunteered from external users as a result of these interactions to improve small parts of the Kokkos implementation. These latter two points are a sign that the community is growing and more actively participating in advancing Kokkos, a necessary step for a more sustainable future where users may answer other users questions, and the community begins to provide new features and solutions to problems.

Next Steps There are two main thrusts of development underway: the writing of a Kokkos API Reference and the development of more advanced tutorials. While the Kokkos Programming Guide and the Tutorial Presentations are well received, more advanced users often only want to look up the API of Kokkos features. Such an API reference is not yet available. Its development is under way, and we hope to have it cover the majority of Kokkos features by the end of Summer 2018.

For tutorials, feedback provided by attendees indicate that some of the material covered in the standard tutorial is a bit too advanced for an introduction to Kokkos. On the other hand, as the number of users who have had previous exposure to Kokkos is growing, they are asking for more of the advanced features to be covered. To support both types of attendees, it is clear that splitting the tutorial into beginner and advanced sections, as well as extending the advanced section beyond what is currently covered is necessary.

Finally, we would like to improve the accessibility of all of the resources that are being developed to support Kokkos and increase its adoption. Currently, all documentation, tutorials, references, and support channels are in various locations that are best suited to their differing format requirements. However, having a one-stop landing page where new encounters can learn about the project and current users can find the location of all available resources would increase usage of the various materials and communication channels within the community, among both developers and users.

4.1.12 2.3.1.11 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% [41]) and the majority of software technology (ST) projects (57% [41]) 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.

Without the OMPI-X project, there will be less competition and less innovation in addressing the needs of ECP users in the critical area of scalable, performant, and capable exascale-quality inter-process communication capabilities.

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. The MPI ecosystem (the standard and its implementations) needs to evolve to address challenges driven by architectural change, as well taking advantage of new features and capabilities. As applications and libraries build up to exascale, the number of node, processes, and threads required will rise significantly, whereas other key resources, such as memory tend to go *down* on a per-node, -process, or -thread basis. This emphasizes the importance of scalability in terms of both performance and resource utilization. Finally, we must work with in the much larger and broader MPI community to find approaches to address these challenges which do not adversely impact the capabilities, performance, or scalability for other users of MPI and Open MPI.

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 will develop APIs and demonstrate capabilities for interoperability in both MPI+X and multiple inter-process runtime situations.

Extending the MPI Standard to Better Support Exascale Architectures The MPI community is considering for standardization a number of ideas that are particularly important to supporting the architectural and system size characteristics anticipated for exascale. "Finepoints" and "Endpoints" 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. We will develop prototype implementations and work with ECP teams to evaluate the ability of these approaches to address ECP requirements in order to facilitate the standardization process.

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 in order to improve scalability. OMPI-X is targeting memory usage within Open MPI, as well as 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. We will provide implementations of the User-Level Fault Mitigation (ULFM) and ReInit proposals currently under discussion within the MPI Forum, as well as demonstrations of their use, in order to help drive standardization discussions, and to help ECP team understand how they can take advantage of these capabilities to improve the resilience of their libraries and applications.

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.

Recent Progress One of the first activities of the project was to conduct a survey of ECP AD and ST projects to better understand their current and expected usage of MPI, and look for significant issues in the MPI environment that we might have overlooked in our planning for the project. A paper describing the survey results was presented at the ExaMPI 2017 workshop held in conjunction with SC17 [41], and will appear in a special issue of Concurrency and Computing: Practice and Experience. The survey has generated a great deal of interest in the MPI community, and there are plans to expand the scope to provide a better characterization of MPI usage around the world.

We have delivered an implementation of the User-Level Fault Mitigation (ULFM) resilience approach, which are under consideration by the MPI Forum for inclusion in the standard. ULFM provides the basic building blocks for cheap, tailored recovery capabilities within applications and libraries using MPI. ULFM imposes no overhead on raw communication performance on ECP-relevant hardware. We are now working with several application teams to demonstrate the capabilities it provides.

We expect to be working on scalability and performance of Open MPI throughout the project, but some early successes have been demonstrated. We have improved the RMA implementation so achieve performance levels comparable to those obtained only by high tuned implementations by vendors and significantly improved their performance in multi-threaded contexts (Fig. 20). We have also been able to improve message matching by up to 2× generally, and up to 45× on Intel Xeon Phi processors, and we have made significant improvement to performance when MPI is used in a multi-threaded environment.

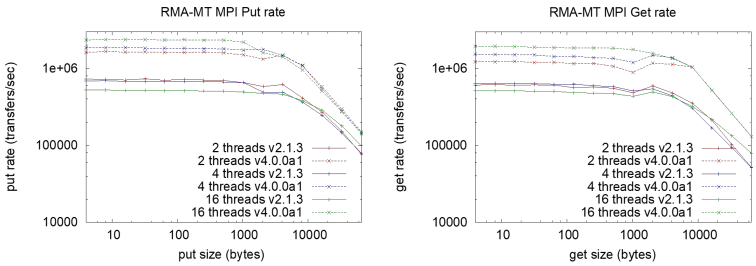


Figure 20: Comparison of put (left) and get (right) RMA performance in a multi-threaded context for Open MPI. Recent OMPI-X contributions are reflected in version 4.0.0a1 (top group of lines), in comparison with v2.1.3.

Early work with the prototype implementation of Finepoints shows improvements of 25% in communication costs and 5% in overall execution time for one ECP mini-app. Open MPI support for the MPI_T interface has been extended [42] to provide a set of low-level counters to present a more detailed performance characteristics map to tools and to users. Finally, we have deployed the MTT testing infrastructure used by Open MPI on ORNL's SummitDev and Summit platforms, as well as improving the MTT system itself.

Next Steps We are making progress across multiple fronts, some of which has been described above. In FY18, we expect to complete our primary efforts in the area of resilience in Open MPI, an implementation of the "ReInit" approach to complement the already completed User-Level Fault Mitigation (ULFM) capability. Both approaches continue to be debated within the MPI Forum, but we are hopeful that demonstrations based on our implementations can help bring these discussions to a conclusion. Early in FY19, we will also be delivering a formal proposal and prototype implementation of the "Finepoints" approach to supporting environments with large numbers of threads.

4.1.13 Runtime System for Application-Level Power Steering on Exascale Systems

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

The ECP PowerSteering project is developing a *job-level* power management runtime system that will optimize performance of Exascale scientific applications transparently under power and/or energy constraints. Existing research efforts, including Conductor and Adagio, are being actively integrated into Intel's GEOPM runtime system, an ongoing open source effort led by Intel. This integration expands GEOPM?s capabilities with the latest research while providing a production-grade, industry-supported open source solution. By developing new platform plugins, this project also supports upcoming target platforms and paradigms for ECP beyond the Intel architectures, and incorporates task-based programming models such as Legion. By being both configurable and cross-platform, GEOPM will help applications achieve maximum performance under a power constraint.

This project is essential for ECP because it enables Exascale applications to operate safely with optimal performance under power and energy constraints. This project is also essential for building a sophisticated hierarchical software stack proposed by the ECP Argo and ECP Flux projects. Additionally, the project fulfills an essential need for ECP by enabling vendor and academic collaborations that provide for accelerated adoption of best practices and better interoperability at scale. By leveraging the GEOPM software developed in this project, compute centers can safely operate under power and energy constraints while maximizing performance and scientific throughput.

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

Another set of challenges come from portability issues. Existing solutions are targeted toward specific Intel microarchitectures as well as programming models. Additionally, some of the existing solutions violate the specified power budget before reaching a steady state, resulting in power fluctuations as well as unsafe operation. As part of this project, we strive to provide portability as well as safe operation using both hardware-level and application-level information for adaptive configuration selection and critical path analysis.

Solution Strategy Our solution is to develop a job-level runtime system (Intel GEOPM) that can operate transparently to user applications, and can also cooperate with HPC resource managers and node-level tools. We are taking a two-pronged approach. First, we are working toward consolidating existing research efforts from the community to develop high-quality plugins for GEOPM that can be deployed at Exascale. In parallel, we are developing new algorithms in GEOPM to address other Exascale challenges such as heterogeneity and variation. While GEOPM already provides some baseline algorithms, the existing capabilities are not programmer transparent and not sufficient for Exascale. Our advanced algorithms analyze critical paths of scientific applications transparently, balance power between different components intelligently, and provide mechanisms to capture fine-grained application semantics through Caliper. Additionally, these advanced algorithms will support non-Intel architectures such as IBM/NVIDIA and novel task-based programming models such as Legion, which are critical for portability in the future. We also intend for GEOPM to be a part of a holistic power management stack that does dynamic, hierarchical power management and works closely with resource managers such as SLURM or Flux. In order to accomplish portability and smooth integration, we are closely collaborating with ECP Argo and ECP Flux projects, with University of Arizona, and with Intel and IBM.

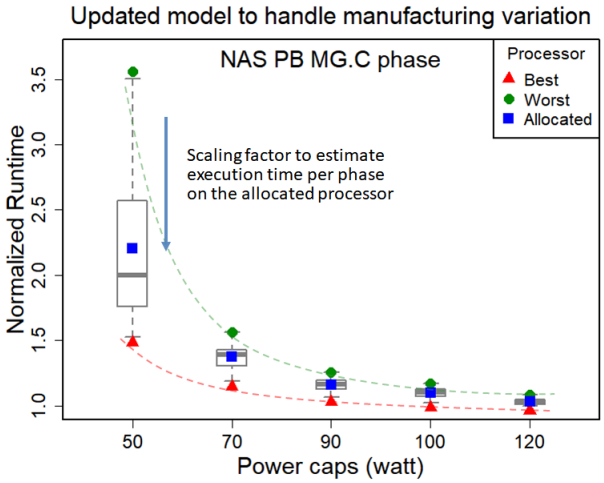


Figure 21: Non-linear power-performance model in use for MG.C during configuration exploration phase for the runtime system

Recent Progress Recently, we achieved two milestones in March 2018. The first was to update the power model for our plugin to incorporate application phases and manufacturing variation, and the second milestone was to support task-based programming models in GEOPM. We developed an offline power/performance model based on processor characterization over codes with broad spectrum of compute and memory-boundedness at different processor power caps. We also updated the configuration space exploration to use this model to adjust per-MPI rank performance measurements over each computation phase.

We are now working on testing and evaluation of our framework with the new model and collecting new data on the Quartz cluster at LLNL. Some early results are presented in Figure 21. The figure shows the compute phase of MG.C, where the runtime system uses a non-linear power-performance model during the configuration exploration phase to account for manufacturing variability. For our second milestone, we developed an MPI + Legion + GEOPM interoperability benchmark that allows us to use GEOPM for dynamic power management of task-based models.

Next Steps We will continue our research and development work as planned toward the September 2018 milestones. More specifically, we are working on porting GEOPM to non-Intel architectures (IBM Power8 or Power9, and NVIDIA GPUs are candidates). We will also enhance our variation-aware and phase aware model with advanced machine learning and statistical techniques. We also plan to improve the overhead of the configuration exploration function by selecting configurations that minimize sampling overhead without a significant impact on the prediction accuracy of the power model especially at lower power budgets.

One of our current challenges is to gain access to non-Intel architectures such as IBM Power8/Power9 and NVIDIA GPUs with elevated privileges that are required for power management. We are working with LLNL to gain such access. Additionally, for our Legion work, we are working toward understanding mappers as well as task distribution better in order to determine the spatial and temporal aspects of power management with GEOPM plugins. We are also looking into S3D application code as part of our Legion power model exploration. Lastly, we are looking into adding Spack support for installing GEOPM.

4.1.14 SOLLVE

Overview OpenMP is a directive-based API for shared-memory and accelerator systems. It is supported by a stable community of vendors, research labs, and academics who participate in the efforts of the OpenMP Architecture Review Board; it is also the most used API for intra-node programming in ECP applications. Implementations are available in all DOE LCFs, and a variety of programming tools are available to support OpenMP application development. The mission of the SOLLVE project is to further enhance the OpenMP specification and implementations to meet the performance and productivity goals of ECP applications. We directly interact with DOE end-users in order to understand their application software requirements. We will develop OpenMP solutions for ECP needs; propose features for standardization; produce prototype implementations of new features to support their rapid adoption; develop and deliver a verification and validation (V&V) suite to assess implementations and enable evaluations by DOE facilities, and deliver a high-quality, robust ECP OpenMP implementation in the LLVM compiler framework. Attaining high levels of single-node performance and meeting performance portability needs will only be possible via enhancements to OpenMP in such areas as data motion and placement in complex memory hierarchies, efficiency in the context of C++, and features that allow the creation of performance portable code. SOLLVE plays a critical role in identifying, implementing, promoting, and deploying key functionality that will enable ECP application developers to reach their goals using OpenMP. We will demonstrate the high impact of new features via their use in selected ECP applications.

Key Challenges The SOLLVE project addresses a number of key challenges faced by DOE facilities, application groups and scientists. The first of these is the gap between existing OpenMP functionality and user requirements. This problem is largely exacerbated by ever increasing complexity and heterogeneity of computing systems. The second challenge is to suitably evolve the OpenMP specification to satisfy the identified requirements. This process involves coordinating with vendors and other members of OpenMP's language committee to reach consensus on the scope of the API, syntax and semantics of new features. The third challenge addressed by the SOLLVE project is performance portability. Current hardware and software trends [43] have shown that pre-Exascale and Exascale systems will be extremely heterogeneous, and may consist of diverse architectures such as CPUs, GPUs, FPGAs, among others. Programmatically, this means that memory hierarchies will be even deeper, and multiple levels and types of parallelism will have to be exposed, extracted and mapped onto these systems to achieve suitable performance levels. This naturally places heavy emphasis on application readiness, where APIs such as OpenMP will need to address performance portability to serve the critical role of abstracting hardware and software complexities. Finally, the last challenge is to fully assess the quality of delivered OpenMP implementations, their compliance with the specification and potential divergence that, if unidentified, could can lead to undesired program behavior.

Solution Strategy The challenges discussed here are addressed by 5 major thrust areas, which we describe below as part of the SOLLVE process:

1. **Application requirements** The first step of the SOLLVE process consists in collecting user requirements from relevant and representative ECP applications (e.g. QMCPACK or Lattice QCD). Collected requirements are then converted to use-cases which are then handed to the OpenMP Standard Committee.

2. **OpenMP specification evolution** This thrust area is responsible of translating use-cases to new OpenMP features to be introduced into the standard. In this phase, the semantics of the new capabilities are defined and formalized, followed by early prototypes in one or more OpenMP implementations (either vendor or open source solutions). Broadly, the bulk of new features fall into the areas of accelerator, affinity, tasking or memory management.

3. **Lightweight OpenMP runtime** SOLLVE is committed to deliver an open source, lightweight and scalable OpenMP implementation (the BOLT runtime) that will fulfill ECP needs. This deliverable serves two key objectives: i) early access to a stable implementation; and ii) risk mitigation.

4. **LLVM** This compiler infrastructure is ideal for delivering high-quality and deployable OpenMP implementations. Our effort enhances the LLVM framework by introducing compiler transformations

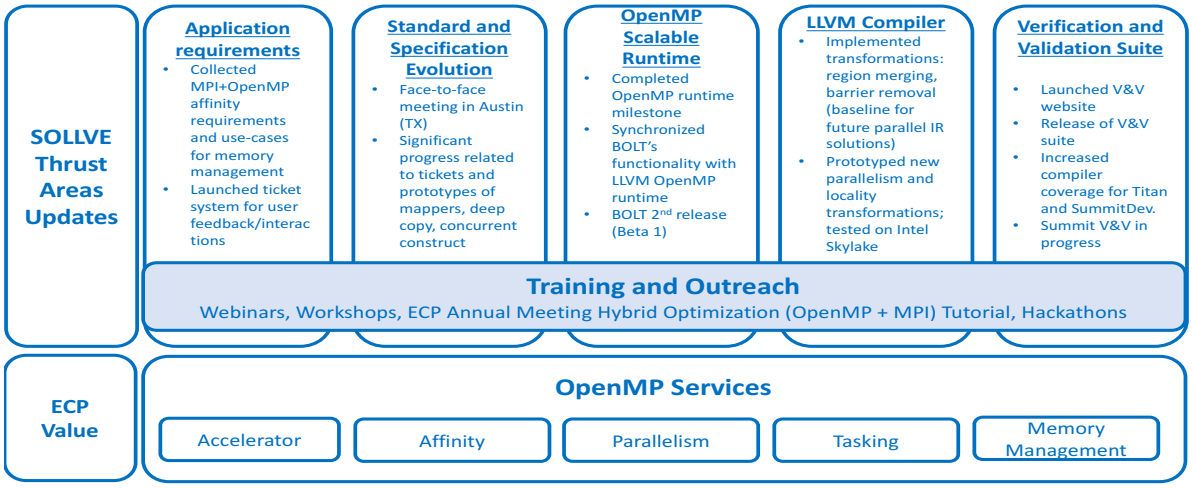


Figure 22: SOLLVE thrust area updates

that leverage both prototyped OpenMP features and those introduced in OpenMP 4.5. User adoption of new OpenMP capabilities could vary from months to years. Thus, delivering compiler technology that automatically transforms user's code targeting the new OpenMP functionality (e.g. target offloading with data mappers), represents high return value for ECP in terms of time and resources.

5. **Validation and verification (V&V)** This thrust focuses on designing and implementing a benchmark suite that allows to assess the coverage and standard compliance of several OpenMP implementations (LLVM, BOLT, IBM XL, NVIDIA, etc). In addition, a ticket system for bug reporting and inquiries has also been deployed to facilitate interaction with end users.

Recent Progress Figure 22 shows the latest progress on the 5 core SOLLVE thrust areas. We note that the **training and outreach** activity is a cross-cutting effort which includes resources from the SOLLVE project and external partners, namely collaborators from Lawrence Berkeley National Laboratory, Oak Ridge, University of Delaware and other academic institutions. In addition to the above updates, a number of articles have also been published as part of the SOLLVE effort [44, 45, 46, 47].

Next Steps As next steps planned for SOLLVE we have

- Applications: gather more requirements for memory management API and concurrent parallel construct; prepare and coordinate OpenMP webinar focusing on memory management, deep copy and tasking.
- OpenMP standard: Next face-to-face meeting in May; ratify and vote latest memory management features, mappers and parallelism descriptors
- OpenMP runtime: design and implement high-level interface for passing scheduling/blocking hints
- LLVM compiler: develop new optimizations leveraging prototype parallel-IR; refine Clang based implementation of data layout transformations; evaluate new compiler transformations on Intel KNL and ARM architectures
- V&V suite: identify performance critical kernels from selected applications; prepare release for SC18; coordinate suite deployment with CORAL systems.

4.1.15 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 co-located 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 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 as powerful or flexible. Existing runtimes tailored for generic use [48, 49, 50, 51, 52, 53, 54, 55, 56] are suitable as common frameworks to facilitate portability and interoperability but offer insufficient flexibility to efficiently capture higher-level abstractions, while specialized runtimes [57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67] are oriented to a 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. Argobots honors this high degree of expressibility through three key aspects:

1. Argobots distinguishes between 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. Work units execute within OS-level threads, which we refer to as *execution streams* (ESs). Unlike existing generic runtimes, ESs are exposed to and manageable by users.

3. Argobots allows full control over *work unit management*. Users can freely manage scheduling and mapping of work units to ESs and achieve the desired behavior.

In order to ensure fast critical paths despite the rich set of capabilities, Argobots was designed in a modular way to offer configuration knobs and a rich API that allow users to trim unnecessary costs.

Recent Progress The proposed lightweight, low-level threading and tasking framework that is designed as a portable and performant substrate for high-level programming models or runtime systems is called as Argobots. The design of the Argobots is as follows:

1. **Execution Model:** Figure 23 illustrates the execution model of Argobots. Two levels of parallelism are supported: ESs and work units. An ES maps to one OS thread, is explicitly created by the user, and executes independently of other ESs. A work unit is a lightweight execution unit, a ULT or a tasklet, that runs within an ES. There is no parallel execution of work units within a single ES, but work units across ESs can be executed in parallel. Each ES is associated with its own scheduler that is in charge of scheduling work units according to its scheduling policy.

2. **Scheduler:** Argobots provides an infrastructure for stackable or nested schedulers, with pluggable scheduling policies, while exploiting the cooperative non-preemptive activation of work units. Localized scheduling policies such as those used in current runtime systems, while efficient for short execution, are unaware of global policies and priorities. Argobots allows each ES to have its own schedulers. To execute work units, an ES has at least one main scheduler (S_M). A scheduler is associated with one

or more *pools* where ready ULTs and tasklets are waiting for their execution. Stacking schedulers is achieved through pushing schedulers into a pool. When a higher-level scheduler pops a scheduler from its pool, the new scheduler starts its execution (i.e., scheduling). Once it completes the scheduling, control returns to the scheduler that started the execution. To give control back to the parent scheduler, a scheduler can also yield. To support plugging in different scheduling policies, all schedulers, including the main scheduler, and pools are replaceable by user-provided alternatives.

3. **Primitive Operations:** Argobots defines primitive operations for work units such as **Creation**, **Join**, **Yield**, **Yield_to**, **Migration** and **Synchronizations**. Since tasklets are used for atomic work without suspending, most operations presented here—except creation, join, and migration—apply only to ULTs.

Argobots is implemented in C language. An ES is mapped to a Pthread and can be bound to a hardware processing element (e.g., CPU core or hardware thread). Context switching between ULTs can be achieved through various methods, such as `ucontext`, `setjmp/longjmp` with `sigaltstack` [68], or Boost library's `fcontext` [69]. A pool is a container data structure that can hold a set of work units and provides operations for insertion and deletion. Argobots relies on cooperative scheduling of ULTs to improve resource utilization. That is, a ULT may voluntarily yield control when idle in order to allow the underlying ES to make progress on other work units.

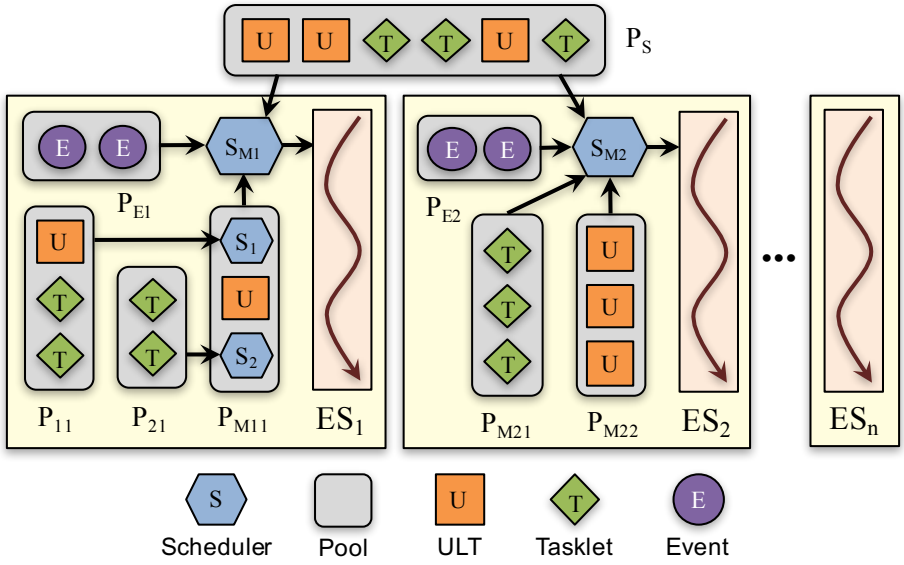


Figure 23: Argobots execution model

Next Steps We studied three use cases of Argobots for higher level runtimes as follows:

1. **OpenMP over Argobots:** Our OpenMP runtime implemented over Argobots (called BOLT) outperforms other OpenMP implementations because of using lightweight work units. We plan to leverage the supported features.

2. **Interoperability with MPI:** We investigated an MPI runtime that interoperates with Argobots ULTs instead of OS-level threads. The runtime has been shown to be subject to lock management issues directing us to further work on lock implementation.

3. **Colocated I/O services:** The most straightforward way to utilize Argobots within an I/O service daemon is to create a new ULT to service each incoming I/O request. We implemented two small extension libraries to help support this use case. The first, *abt-io* and the second is *abt-snoozer*. Further the service routines are decomposed into smaller discrete event-driven routines with disjoint stacks, a technique known as *stack ripping* [57]. Presently, Argobots significantly reduce the development, debugging, and maintenance burden for system services and needs further investigations.

4.1.16 BOLT: Lightning Fast OpenMP

Overview OpenMP is central for several applications that target Exascale, including ECP applications, to exploit on-node computational resources. Unfortunately, current production OpenMP runtimes, such as those that ship with Intel and GNU compilers, are inadequate for the massive and fine-grained concurrency expected at the Exascale level. These runtimes rely on heavy-handed OS-level threading strategies that incur significant overheads at fine-grained levels and exacerbate interoperability issues between OpenMP and internode programming systems, such as MPI and OpenSHMEM. Our solution is a production quality OpenMP runtime (called BOLT) that leverages user-level threads instead of OS-level threads (e.g., Pthreads). Due to their lightweight nature, managing and scheduling user-level threads incurs significantly less overhead. 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 [70, 71]. 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 explicit tasks. Nesting 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 compels 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. For instance, when OpenMP threads are allowed to concurrently perform internode communication, the interoperability layer dictates how thread safety and progress on communication is handled. It is notoriously known that OS-level thread safety in production MPI libraries suffers significant performance issues. Despite the recent improvements to OS-level thread safety in MPI libraries, the state-of-the-art performance results are still indicating serious scalability issues. While continues 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 are being investigated to allow reducing thread management overheads even further. Interoperability improvements have also been demonstrated by having BOLT interoperate with an MPI library (MPICH) through the Argobots threading layer rather than OS-level threads. Results showed that this approach allows better communication progress and outperforms the traditional Pthreads-level interaction.

Recent Progress The development of BOLT went through several steps that involved developing or optimizing various aspects. The first step was to fork BOLT from the upstream LLVM OpenMP runtime (https://openmp.llvm.org) and replace the Pthreads layer with a basic Argobots design. Next, we implemented advanced scheduling strategies to improve the performance of BOLT under fine-grained nested parallelism

regimes. Figure 24 shows the development of BOLT. The following step was to investigate ways to reduce thread management overheads by exploiting application knowledge. By taking into account the suspension likelihood of a thread or task (e.g., yield execution on a blocking I/O operation), we have extended Argobots to leverage such information and incorporated these features into the BOLT runtime. With respect to interoperability with MPI, we have investigated the shortcomings of current interoperability models at the implementation as well as standard specification levels. We have investigated the benefits of having a more lightweight interoperability layer through Argobots using BOLT as the OpenMP runtime and MPICH as the MPI library and demonstrated encouraging results. BOLT and Argobots have been subject to two releases and our latest progress results have been published in the prestigious IEEE Transactions on Parallel and Distributed Systems journal and some results have been described in an ECP report.

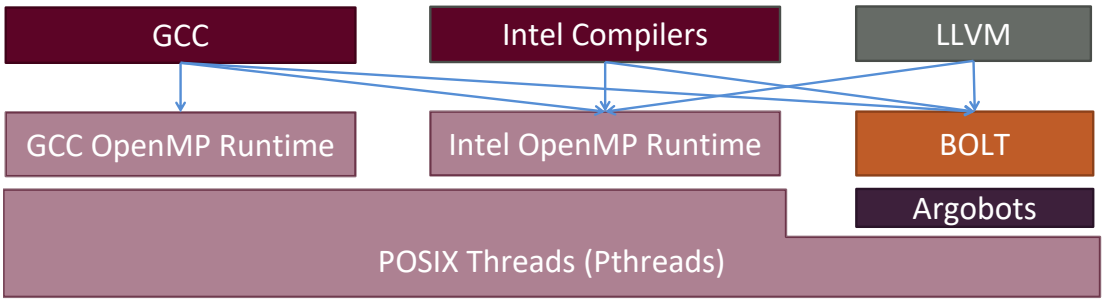


Figure 24: Pictorial representation of development of BOLT

Next Steps Some aspects of BOLT require further work and investigation. In the following we enumerate some of the key aspects under progress:

1. User hints to lower thread management overheads previously described were only implemented at the runtime level. We plan to work with the compiler team of the SOLLVE project to integrate these hints at the compiler level.

2. Only benchmarks and a few application cases have been used for evaluation purposes. We plan to investigate a wider range of applications, including ECP ones, to evaluate the effectiveness of our solution and potentially discover new areas of needed improvement.

3. The interoperability results between BOLT and MPI were encouraging but still preliminary. We are planning on doing more extensive evaluation of this interoperability layer with benchmarks and applications and potentially implement improvement if scalability is unsatisfactory.

4. Finally, we plan to perform more rigorous robustness testing with validation and test suites as well as with fully-fledged applications and proxy-application for as much coverage as possible.

4.1.17 UPC++

Overview The UPC++ project is developing a C++ library that supports Partitioned Global Address Space (PGAS) programming [72]. The UPC++ project began in 2012 with a prototype designated V0.1, described in [73]. We are revising the library under the auspices of the DOE's Exascale Computing Project, to meet the needs of applications requiring PGAS support. UPC++ is well-suited for implementing elaborate distributed data structures where communication is irregular or fine-grained. The UPC++ interfaces for moving non-contiguous data and sending Remote Procedure Calls (RPC) are composable and closely resemble those used in modern C++.

UPC++ is needed for ECP because it delivers low-overhead communication that runs at close to hardware speeds, embracing interest by vendors in the PGAS model because it efficiently matches the RDMA mechanisms offered by network hardware and on-chip communication between distinct address spaces. Because ECP applications rely on irregular representations to improve accuracy and conserve memory, the UPC++ library provides an essential ingredient for the ECP software stack. It will enable effective scaling in Exascale software by minimizing the work funneled to lightweight cores, avoiding the overhead of long, branchy serial code paths, and supporting efficient fine-grained communication. The importance of these properties is exacerbated by application trends; many ECP applications require the use of adaptive meshes, sparse matrices, dynamic load balancing, or similar techniques. UPC++'s low-overhead communication mechanisms can maximize injection rate and network utilization, tolerate latency through overlap, streamline unpredictable communication events, minimize synchronization, and efficiently support small- to medium-sized messages arising in such applications. UPC++ will enable the ECP software stack to exploit the best-available communication mechanisms, including novel features being developed by vendors. This library offers a complementary, yet interoperable, approach to MPI with OpenMP, enabling developers to focus their effort on optimizing performance-critical communication.

Key Challenges As the result of technological trends, the cost of data motion is steadily increasing relative to that of computation. To reduce communication costs we need to either reduce the software overheads or hide communication behind available computation. UPC++ addresses both strategies. To reduce software overheads, UPC++ takes advantage of the GASNet-EX communication library's [74] low-overhead communication as well as access to any special hardware (see the accompanying report on GASNet-EX, which is being co-designed). UPC++ supports asynchronous communication via classic one-sided communication (i.e. puts and gets) and remote procedure calls, to support communication hiding.

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

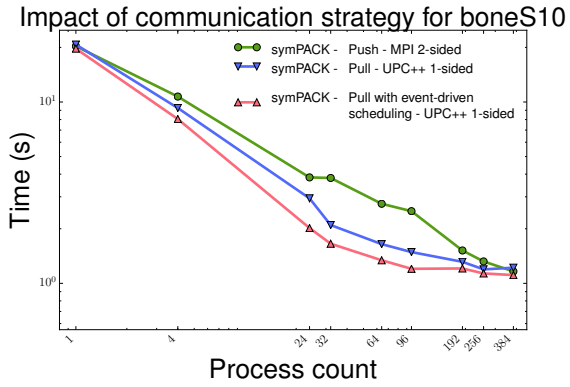
1. **Increased performance through reduced communication costs:** The UPC++ programmer can expect communication to run at close to hardware speeds. Asynchronous execution enables an application to hide communication behind available computation.
2. **Improved productivity:** UPC++'s treatment of asynchronous execution relies on futures and promises, and these simplify the management of asynchrony.

The PGAS one-sided communication employed by UPC++ (get/put) benefits application performance by mapping tightly onto the RDMA communication supported by the communication network. 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 synchronization overheads of two-sided communication (e.g. message passing).

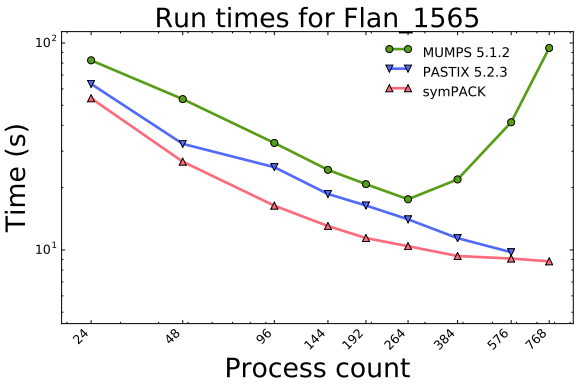
UPC++'s Remote Procedure Call, which is built on GASNet Active Messages, provides additional control over asynchronous execution, by enabling the programmer to execute procedure calls on remote processors. RPC is useful in managing access to complicated irregular data structures, and in expressing asynchronous task execution.

UPC++ addresses productivity via one-sided data motion, remote procedure calls, and via the provision of futures. Futures enable the programmer to capture data readiness state, which is useful in making scheduling decisions, via continuations, to execute asynchronously as dependencies become satisfied. Chaining and conjoining of asynchronous operations simplify treatment of their completion.

Recent Progress symPACK is a direct linear solver for symmetric positive definite sparse matrices. Originally written using the legacy UPC++ V0.1, we have recently ported symPACK to use the latest release of UPC++ V1.0. Our experiments conducted on NERSC Edison confirm that the new UPC++ version preserves the performance of the symPACK solver.



(a) Push – MPI two-sided communication
Pull – UPC++: RPC + RMA Get when ready
2 variants with and without event driven scheduling



(b) Strong scaling of symmetric solvers (Factorization time only)

Figure 25: Performance of the symPACK solver using UPC++ V1.0

The first experiment, depicted in Fig. 25a compares the performance of three implementations of symPACK:

- a **Push** strategy (the sender "pushes" the outgoing message as soon as possible) using non-blocking MPI two-sided messages,
- a **Pull** strategy (sender "notifies" with RPC, receiver RMA gets when ready) using one-sided UPC++ RPC and RMA get,
- the same **Pull** strategy using one-sided UPC++ RPC and RMA get, combined with an event-driven dynamic scheduling policy.

UPC++ allows symPACK to implement an efficient Pull strategy using one-sided operations in a simple and efficient fashion. This implementation surpasses the performance of the original MPI two-sided variant. Fig. 25b compares the strong scalability, on NERSC Edison, of symPACK with other state-of-the-art solvers for sparse symmetric matrices. symPACK can be seen to significantly outperform the other solvers on this particular problem, a trend which can be observed on most matrices from the SuiteSparse matrix collection.

Next Steps Our next efforts are:

1. **Team-aware APIs:** Teams are a mechanism for grouping ranks. Team-aware APIs will be developed to enable teams to be used not only in collective communication (see below) but also in distributed objects, and certain modes of accessing global shared storage.
2. **Support for non-blocking collectives:** UPC++ supports a small number of collectives, and this set will be expanded to accommodate the needs of our application partners. Support for teams will also be included.

4.1.18 GASNet-EX

Overview The Lightweight Communication and Global Address Space Support project (Pagoda) is developing GASNet-EX, a portable high-performance communication layer supporting multiple implementations of the Partitioned Global Address Space (PGAS) model, including Pagoda's PGAS programming interface UPC++ [72] and the Legion Programing System [75, 76] (WBS 2.3.1.08).

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

We are co-designing GASNet-EX with the UPC++ development team with additional input from the Legion and (non-ECP) Cray Chapel [77, 78] projects.

Key Challenges Exascale systems will deliver exponential growth in on-chip parallelism and reduced memory capacity per core, which will increase the need for strong scaling and thus smaller communication events. To scale well, Exascale software needs to minimize the work performed by lightweight cores and avoid the overhead of long, branchy serial code paths; it needs to support 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 point to the need for low-overhead communication mechanisms that can maximize injection rate and network utilization, tolerate latency through overlap, accommodate unpredictable communication events, minimize synchronization, and efficiently support small- to medium-sized messages. The ECP software stack needs to expose the best-available communication mechanisms, including novel features being developed by the vendor community.

Solution Strategy The Global Address Space (GAS) 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 *GAS* for models that support one-sided communication (including contiguous and non-contiguous remote put/get operations and atomics). Many of these models also include support for remote function invocation. GASNet-EX is a communications library that provides the foundation for implementing GAS models, and is the successor to the widely-deployed GASNet library. We are building on 15 years of experience with the GASNet [74, 79] communication layer to provide production-quality implementations that include improvements that address technology trends and application experience.

The goal of the GASNet-EX work is to provide a portable, high-performance GAS communication layer for Exascale and pre-Exascale systems, to address 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. Interfaces for atomics and collectives are being developed to enable offload to current and future network hardware with corresponding capabilities. These design choices and their implementations provide the low-overhead communications mechanisms required to address Exascale applications' requirements.

Recent Progress The 2017.6.0 release of GASNet-EX introduced three new features, known as "New Active Message Interfaces", "Immediate Operations" and "Local Completion". The 2017.12.0 release introduced a new "Remote Atomics" feature, as well as "Expanded VIS Interfaces". These two releases contain network-independent "reference" implementations of these features, which provide implementations in terms of the pre-existing functionality available in GASNet-EX on all networks. While these reference implementations are correct and functionally complete, they do not take advantage of network-specific mechanisms.

The most recent work, present in the 2018.3.0 release, provides network-*specific* implementations of GASNet-EX's new features for the Cray Aries network used in Cray XC-series systems. Performance gains achieved include (1) Negotiated-Payload Active Messages improve bandwidth of a ping-pong test by up to 14%, (2) Immediate Operations reduce running time of a synthetic benchmark by up to 93%, (3) non-bulk RMA Put bandwidth is increased by up to 32%, (4) Remote Atomic performance is 70% faster than the

reference on a point-to-point test and allows a hot-spot test to scale robustly, and (5) non-contiguous RMA interfaces see up to 26% speedups for an inter-node benchmark. More complete details are available in an LBNL Technical Report [80].

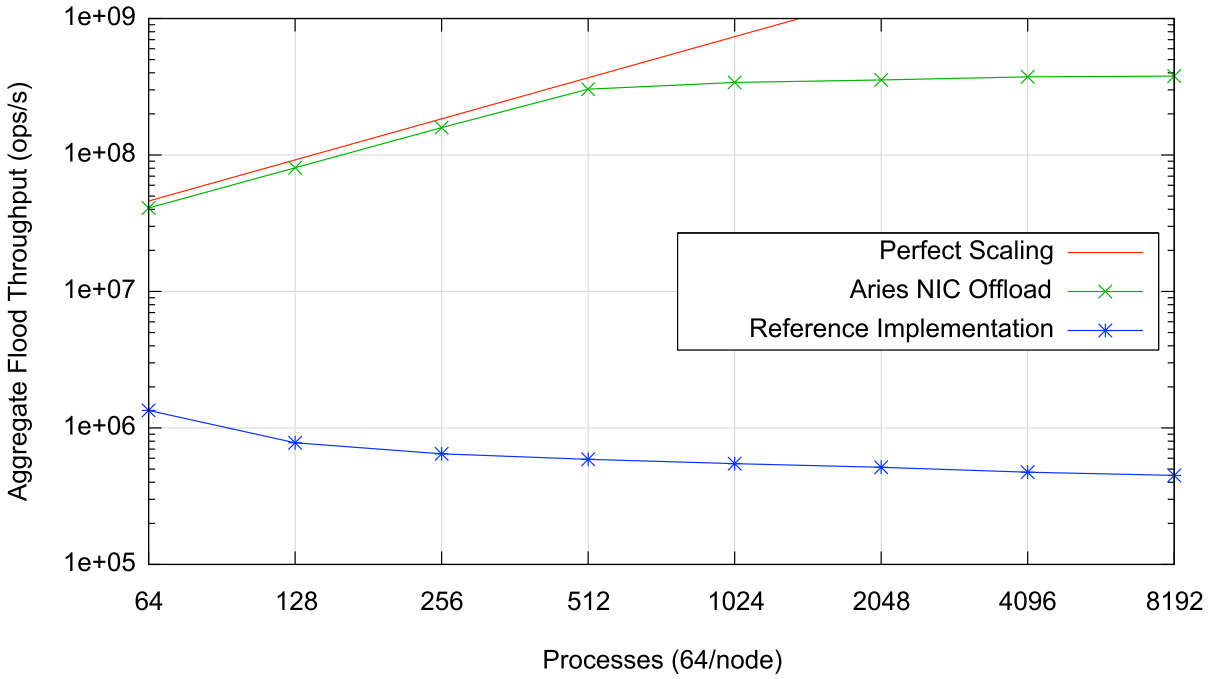


Figure 26: Weak Scaling of 64-bit Unsigned Integer Atomic Hot-Spot Test on ALCF's Theta

The Aries-specific implementation of remote atomics offloads most operations to the Aries NIC, yielding a 1.7x reduction in latency, and greatly improved scalability in a many-to-one atomics hot-spot test. Figure 26 shows results on ALCF's Cray XC-40, Theta, for such a benchmark in which all 64 cores on one or more compute nodes simultaneously perform 64-bit unsigned integer `fetch-and-add` operations on a single location. The figure shows the aggregate throughput as a function of the number of processes. The data shows that as the process count increases, the aggregate performance of the reference implementation actually drops (due to overheads of message reception dominating). Meanwhile the performance of the Aries-specific version rises steadily as the node count increases from 1 to 8 (64 to 512 processes), and continues to rise gradually from that point to the highest concurrency measured (128 nodes = 8192 processes). For comparison, the "Perfect Scaling" line (in red at the upper-left of the figure) shows the throughput of a single-process run scaled by the process count.

Next Steps Our next efforts include:

1. **"Teams and Collectives"** are to be updated (both specification and implementation) for new GASNet-EX interfaces to improve scalability and to enable future offload efforts.

2. **"Dependent Operations"** are to be implemented, to permit client runtimes to express ordered sequences of operations to be executed by GASNet-EX.

4.1.19 Enhancing Qthreads for ECP Science and Energy Impact

Overview "Enhancing Qthreads for ECP Science and Energy Impact" is a project that aims to improve the performance of applications that use multithreading with communication, e.g., MPI. Most ECP applications are using this combination of programming models, with the Kokkos or RAJA performance portability libraries and/or the OpenMP API for multithreading. This project supports the Kokkos ECP software project and OpenMP from the underlying runtime layer to deliver thread-scalable performance to those applications. To that end, our projects is developing techniques to incorporate support for better network concurrency into the multithreading runtime system.

Key Challenges Our project addresses the challenge of scalably coupling multithreaded parallelism on the many-core node with communication such as MPI, which has traditionally performed poorly in multithreaded mode. The key challenge arises when multiple threads make communication calls, and those calls must be serviced by the MPI implementation and NIC. Existing solutions, such as MPI_THREAD_MULTIPLE, are often plagued by synchronization overheads. Even the best vendor MPI implementations incur high overheads when the number of threads exceeds the number of hardware contexts in the NIC. While current mechanisms are insufficient even for today's systems, emerging interconnect technologies expose even more network parallelism that must be exploited to maximize performance for Exascale.

Solution Strategy Unlike previous approaches, we attack the problem not only from the communication side (MPI), but with assistance from the multithreading runtime system. Our work adds capabilities to enable the runtime system to identify and optimize for tasks that use communication, distinct from tasks that perform only local computation. We use the Qthreads runtime [81], a scalable, event-driven library for node-level task parallelism, to implement our solution. This work requires cooperation with a communication library that can scalably process communications operations coming from the runtime system. For this purpose, we pair Qthreads with the new "FinePoints" library for threaded MPI execution developed in the OMPI-X ECP project.

Developed at Sandia Labs since 2007, Qthreads serves as a back-end for Kokkos and the Cray Chapel language, as well as providing a portable native C API. Complementary to the current ECP project focusing on the coupling of the runtime with communication, development of Qthreads core capabilities is part of the NNSA ASC system software portfolio and has also been part of sponsored vendor collaborations and LDRD projects. In addition, the techniques developed in this project will be the subject of tech transfer efforts to OpenMP and MPI. The project technical lead is chair of the OpenMP Subcommittee on Task Parallelism, and one of the other technical experts on the project is a key contributor to the MPI Forum and the OMPI-X ECP project that is enhancing the open-source Open MPI implementation of MPI for exascale. We are leveraging the work of that project, and synergies between Qthreads and the OpenMP and Kokkos tasking models.

Recent Progress Most recently, we added the optional network task annotation to task definitions in Qthreads, allowing the identification of communication tasks to the runtime system. We also demonstrated successful coupling of Qthreads with the MPI FinePoints library. FinePoints uses a partitioned buffer to collect the contributions of the various tasks executing on the runtime's threads. Using only atomics rather than heavyweight locks keeps overhead costs low compared to existing methods like MPI_THREAD_MULTIPE, and unlike the Endpoints proposal, the MPI rank space does not expand with the use of more threads. We ported FinePoints benchmark code to use Qthreads as the multithreading library instead of OpenMP and compared to the performance of the two configurations, shown in Figure 27. The observed equivalence in performance justifies our use of Qthreads as a proxy for OpenMP, wherein Qthreads can be used for ease of rapid protoyping of new capabilities with eventual tech transfer back to OpenMP. These results also serve as baselines to measure the performance of our further optimizations against. Finally, we made an initial port of the miniGhost stencil mini-app to use FinePoints with Qthreads to confirm portability beyond benchmarks. Tuning of that mini-app is currently work in progress.

Next Steps We will investigate several possible optimizations, either in terms of Qthreads runtime improvements alone, or in conjunction with the OpenMPI implementation in which Finepoints is natively

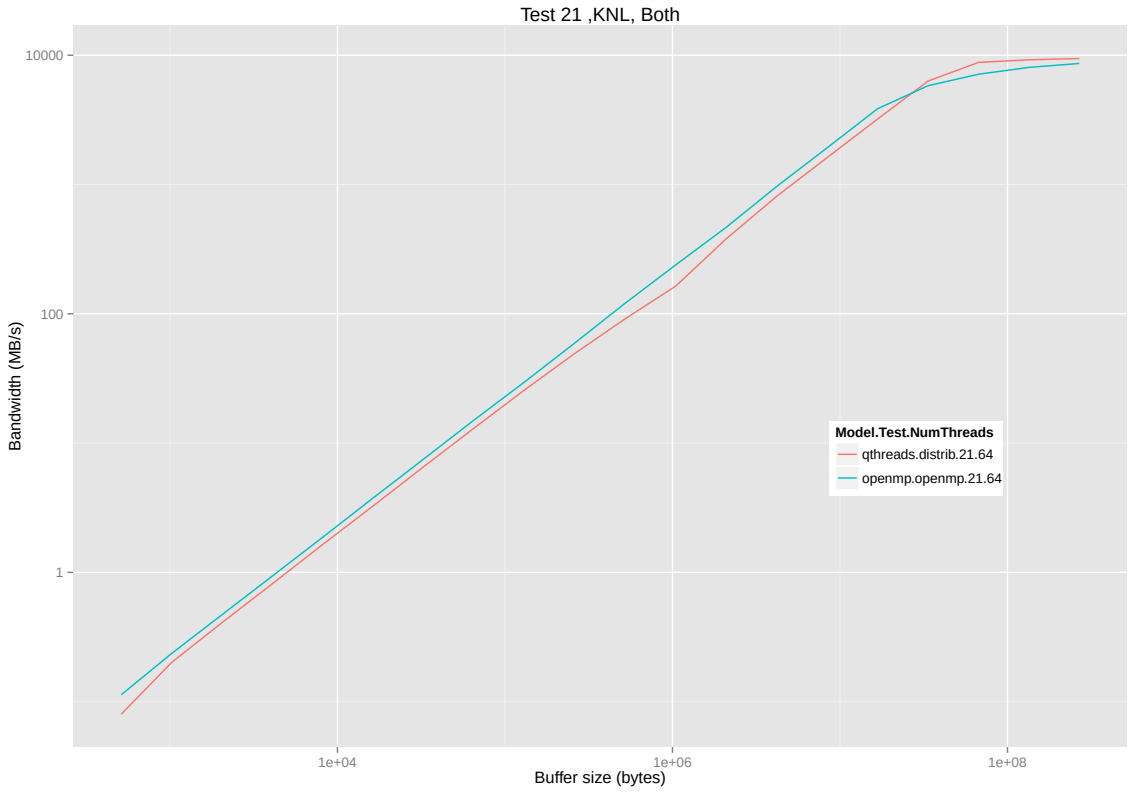


Figure 27: This graph shows the performance of Qthreads and OpenMP paired with the FinePoints library for multithreaded MPI. The x-axis varies the buffer sizes transferred in each experiment in the series, and the y-axis shows the network bandwidth achieved. The similar performance of Qthreads and OpenMP justifies use of the former as a suitable proxy for the latter, with the advantage of flexibility for rapid prototyping of new runtime system techniques.

available as an extension:

- Enabling the swapping of schedulers in the Qthreads runtime at execution time in response to the tasks set observed. (Currently the scheduler must be selected at configure time, so we would progress from there to selection at the start of an execution, and finally to a dynamic selection during the execution.)
- To better enable the coordination of Qthreads and OpenMPI, add Qthreads as an MPI threading model instead of the default pthreads. (This requires modifying OpenMPI's "opal" subsystem to use a new threading mechanism.)
- To explore this scheduling infrastructure, implement novel scheduling points (e.g. synchronization points in critical regions) allowing contended OpenMPI synchronization areas to schedule Qthreads events and vice versa.
- With hierarchical and differential scheduling infrastructure in place, seek to design scheduling policies which optimize the interaction between composed runtimes.

We anticipate that due to technical and logistical constraints, only a subset of these tasks may be possible, but there will be value in at least attempting them and documenting our findings. We feel confident that through this work, significant strides will be made toward better coupling of threading and communication.

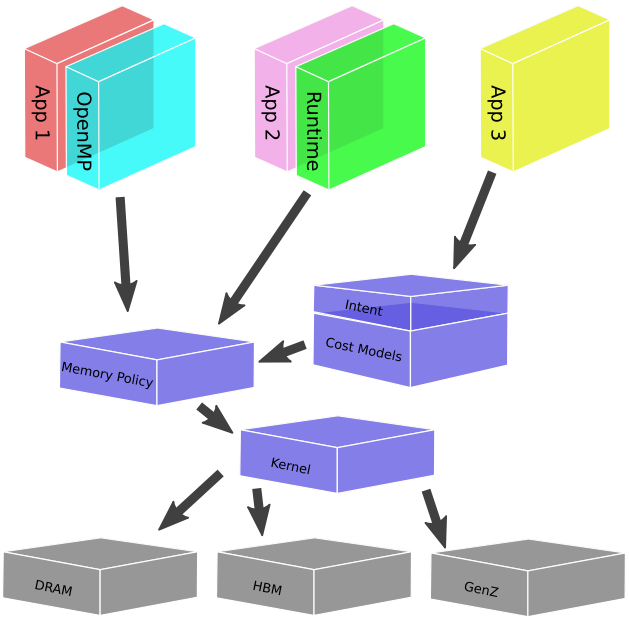


Figure 28: Interface for complex memory that is abstract, portable, extensible to future hardware; including a mechanism-based low-level interface that reins in heterogeneity and an intent-based high-level interface that makes reasonable decisions for applications

4.1.20 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 users 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. The high-level interface starts with a co-design effort involving application developers. It would potentially evolve into more research as intelligent allocators, migrators, and profiling tools are developed. We can achieve success due to our team's extensive experience with runtimes and applications. Our willingness to work with and accept feedback from multiple hardware vendors and ECP developers differentiates our project from existing solutions and will ultimately determine the scale of adoption and deployment.

Key Challenges This effort is a new start and relies on interaction and acceptance of both the vendors and various runtime libraries. Defining a common mechanism to expose various heterogeneous memory devices is important but vendors have not little adopted a standard in this area and will have to be led to a solution.

Solution Strategy The deliverable of this project is the userspace libraries and modifications to the Linux kernel [82] to that would allow better use of heterogeneous memories. The low-level interface would provide support for discovery, allocation/de-allocation, partitioning, and configuration of the memory hierarchy and information on the properties of each specific device (capacity, latency, bandwidth, volatility, power usage, etc.). The interface will be quickly prototyped for a small set of currently available memory technologies (both volatile and non-volatile) and delivered to runtime and application developers, as well as vendors, for feedback on its functionality, performance and features. The high-level interface will leverage the low-level library in order to further decouple applications from hardware configurations. Specifically, it would emphasize ease of use by application developers with a policy/intent driven syntax enabled through run-time intelligence and system support. Applications would specify which attributes (such as fast, low latency, high capacity, persistent, or reclaimable) are a priority for each allocation and the interface will provide the most appropriate configuration. Finally, the missing element is a common set of functions that would constitute a minimal hardware interface to the various types of memory. The goal is to allow fast porting to new hardware but still leave room for innovation by vendors.

Recent Progress

- Low-Level Interface: Progressing on the refactoring of the initial SICM userspace library. Developed the data structure and implementation to track the memory pages in different arenas. High-Level Interface:
- High-Level Interface: Completed initial API design for new high-level SICM persistent memory meta allocator, now called "Metall". Design heavily leverages open-source Boost.Interprocess, and retains some compatibility with it.
- Analysis: Ongoing meetings with Ompi-X team members at Univ. of Tennessee to discuss memory needs and SICM requirements.
- Analysis: evaluating new static analysis capabilities in the Portland Group compiler that generates comprehensive relationship data of both data structures and functions. The results for ACME are stored in a database. As of this quarter, we can now reliably tie trace references back to a line of code via the database.
- Analysis: evaluating ACME using Gem5. We are currently resolving some compatibility issues between the ACME build environment and our Gem5 virtual machine. We now have traces from ES3M and several mini-apps.
- Analysis: Efforts to analyze E3SM and mini-apps using tools by Cray and Intel. These results will be combined with previous results to help characterize barriers and gaps based on feedback from the E3SM team.
- Cost Models: We have extracted a lot of experimental data related to application memory use and layout. This was done with full applications on hardware with the memory throttling-based emulation methodology. Additionally we have finer grained measurements for smaller C benchmarks where we have isolated individual data structures and applied different tiering decisions to assess metrics such as sensitivity, benefit, etc. More of a tool and profiling process to inform the design of the a simple API/API extension that you might integrate with the running application. [83]
- Cost Models: Development of a tool, Mnemo, which provides for automated recommendations of capacity sizing of heterogeneous memories for object store workloads. Given a platform with specific configuration of different memories, and a (representative) workload, we can quickly extract some of the relevant memory usage metrics, and produce cost-benefit estimation curves as a function of different capacity allocations of the different types of memories. The output of Mnemo are estimates which give its users information to make informed decisions about capacity allocations. This can have practical use in shared/capacity platforms, or to rightsize capacity allocations to collocated workloads.

Next Steps

- Low-Level Interface: Finish implementation of refactor and 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. Verify support of emerging OpenMP standards.
- High-Level Interface: Continuing to work on completing first working prototype; performance testing to follow after prototype is complete. Initial performance benchmarks have been selected, including a key-value store with skewed key frequency distribution. These benchmarks also serve as part of a correctness validation test set.
- Document needs of MPI and ACME climate app hybrid memory analysis (with ORNL collaborators and related to UNITY)
- Understand capabilities of hwloc and netloc with respect to OMPI-X needs. Future planned reports will include assessments of what the new SICM tools provide compared to current tools using ACME as a benchmark. The missing gaps are used by the SICM team to inform R/D directions.

4.2 DEVELOPMENT TOOLS

This section present projects in Development Tools.

4.2.1 Development Tools Software Development Kits

Overview The Development Tools 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 4.5.1.

4.2.2 LANL ATDM Tools

Overview The Los Alamos National Laboratory ATDM Tools project provides tools and software infrastructure for improving various aspects of the Exascale programming environment. At present our efforts are focused in two key areas that are prioritized to meet the needs of LANL's ATDM efforts and are also broadly applicable to broader ECP scope:

1. **Kitsune**: An advanced LLVM [84] compiler and tool infrastructure supporting a new explicit parallel intermediate representation for the C/C++ and Fortran programming languages.
2. **QUO**: A runtime library that assists application developers in the composition of multiple software components (e.g. libraries, multi-physics components) that have disjoint mappings to the underlying hardware components. Explicit coordination of this mapping is often critical to not only improve performance but also to make more effective use of computing resources (i.e. allowing developers to avoid over-subscription of a job's node allocations).

Key Challenges The general challenge across both project components is providing and supporting a productive development environment in a currently *yet-to-be-defined* set of system architectures. This is not only difficult given a large number of applications (and application components) but can become increasingly complex if the attributes of potential target platforms are divergent. This requires that we must be flexible and also understand that many platform-centric decisions can have a significant and potentially unexpected impact on our current techniques and software infrastructure.

Solution Strategy Given the project challenges outlined above, our approach takes aspects of today's programming systems (e.g. MPI and OpenMP) into consideration and aims to improve and expand upon their capabilities to address the needs of Exascale computing across a range of application areas. This allows us to attempt to strike a balance between incremental improvements to existing infrastructure along with more aggressive techniques that seeks to provide innovative solutions to help both manage risk and the ability to introduce new breakthrough technologies.

In addition, we are working to form close working relationships with the LLVM community to help provide an impactful and longer term set of technologies to the broader computing community. These activities span both academic and industry collaborations.

Recent Progress For the Kitsune compiler effort, our recent efforts have focused on supporting an infrastructure that maps multiple language constructs from Kokkos, FleCSI and OpenMP into an common intermediate representation that explicitly captures the parallel operations for analysis and optimization (this is a capability that does not exist in the mainline LLVM infrastructure). This parallel representation may then also be targeted to different runtime systems (not necessarily those of use by the initial programming system). See our most recent paper for a discussion of our overall approach [85]. In addition, this work will be presented at the upcoming EuroLLVM workshop [86]. We will continue to use such events to provide the larger LLVM community with updates to our lessons learned and example use cases of this technology.

The QUO infrastructure has been recently deployed and used by LANL's production codes and is in active daily use. Initial steps have been taken to integrate some of its functionality into the Kokkos programming system and we have also briefed code teams at LLNL about the use of the library in multi-physics codes. Our most recent paper on QUO highlights the impact of the library's use on application performance across a range of different applications case studies [87].

Next Steps Both QUO and Kitsune are working towards quarterly milestones and multiple software releases throughout the coming year. At this point in time QUO has reached a production ready state and many activities are focused on performance tweaks, bug fixes and small additions to the overall capabilities. Kitsune is still very much an active proof-of-concept compiler toolchain focused on C and C++ with future plans to add support for Fortran via the Flang project [88]. Even though it is not yet production ready we are actively releasing source code and the supporting infrastructure for deployment as an exploratory and early evaluation candidate.

4.2.3 LLNL ATDM Development Tools Projects

Overview The LLNL ATDM Tools Project came together from two tool efforts at LLNL, ProTools and AID. The ProTools project provides a productization path for research-quality tool software. The ProTools team works with tool research groups and provides the software engineering effort needed to move their tools into production. This includes tasks like writing test suites and documentation, porting to new systems, adding user-driven features, and integrating tools with application codes. Some of the larger efforts in ProTools are:

- Ubiquitous Performance Analysis - A suite of tools and visualizations that enable a performance analysis workflow where tools are built into the application and monitor the performance of every run.
- SCR - The Scalable Checkpoint/Restart library, which abstracts away IO technologies such as burst buffers for applications.
- mpiFileUtils - A suite of IO-tools based on common UNIX file utilities (cp, rm, cmp, ...), but optimized for HPC.
- OMPD - A debugging standardization effort for OpenMP.
- Umpire - An abstraction layer for managing the different types of memory found in current and next-generation systems.

The AID project is developing next-generation debugging and code-correctness tools, with a focus towards tool viability at Exascale The significant projects in AID are:

- STAT - A debugging tool that can narrow down the debugging search space for hangs and other issues at massive scales.
- Archer - A tool for automatically identifying race conditions in OpenMP programs.
- ReMPI/Ninja - A suite of tools that can inject noise in applications to expose MPI races, and record/replay those races when found.
- FLiT - A tool for testing floating point consistency and workloads.

Key Challenges There are several challenging areas that are common not just among the LLNL ATDM Tool Projects, but across tool efforts through-out the HPC community:

- Platform Portability - Tools are particularly challenging to port to new systems. Unlike applications, which rely on standards such as OpenMP and MPI, tools are generally built weaker body of standards. Porting to a new system can involve detailed low-level dives into runtimes and system components. Tools rarely have platform-independent code bases and require significant effort to bring up on new systems.
- Application Complexity - In addition to the increasing complexity of systems, applications are growing in size and complexity. New programming models are increasing the distance between the low-level machine code that tools generally measure and the high-level code that programmers think about. Mapping between these layers requires significant infrastructure in each tool.
- Application Adoption - At any one time it can be easier to add a new printf statement rather than learn a new debugging or performance analysis tool. It has been historically difficult to encourage application teams to consider tool adoption a priority, though with approaching challenges of next-generation and Exascale system this trend seems to be changing.

Solution Strategy Specific solutions to the above problems can vary with each instance and tool. Though there are several high-level trends across the LLNL ATDM Tool Projects:

- Application and Tool Integration - As previously mention, tools can be challenging because they generally live in a smaller standards space. Some efforts are towards shifting tools into the application domain, where they can utilize application-level standards and infrastructure. Examples of this from the Ubiquitous Performance Analysis project and Caliper projects include doing performance attribution

with application-level annotations rather than binary analysis and DWARF line mappings. Or instead of building low-impact tool communication infrastructure like MRNet, we can use MPI have the application team to annotate code with safe communication points.

- Standardization and Co-design - The LLNL ATDM Tool Project is engaging with standards committees to create tool-specific interfaces in programming models. The recent OMPD work added a interface for debuggers into OpenMP, which looks likely to be accepted into OpenMP 5.0. Beyond formal standardization efforts, the LLNL ATDM Tool Projects also work with vendors during system design to ensure the available of tools on upcoming systems. Team members have recently engaged in the CORAL project and are working directly with IBM and NVIDIA on providing numerous tools.
- Tool Componentization and Composability - Rather than have tools reinvent the wheel and then stumble over the same potholes in every system, the LLNL ATDM Tool Projects have striven to make tools that both share common best-practice components and be composedly with other tools. Gotcha, from the Ubiquitous Performance Analysis project, is a new component library that does function wrapping and is being adopted in several other tools. The AID project is building an inter-operable software stack of debugging tools, where projects like STAT and ReMPI can work together with classical debuggers like TotalView and DDT. By sharing code and relying on other tool's strengths we can minimize the amount of repeated work across the tools community.
- Application Adoption - To help application teams adopt tools, members of the LLNL ATDM Tool Projects work directly with them through early tool efforts. This can take the form of software development effort, such as when ProTools team members helped an ASC application adopt SCR. Or it can be a hand-holding exercise when first running a tool, such as when the AID team helped a math library identify a significant race condition with Archer.

Recent Progress There are several recent notable achievements from the LLNL ATDM Tools Projects:

- The OMPD standard looks likely to be adopted into the OpenMP 5.0 standard. In addition to the many OpenMP contributors, The ProTools team worked with the standards group in the LLNL ATDM Data and Visualization project on this. ProTools focused on the reference implementation and the standards group on the standards document.
- The ProTools team will soon be releasing the initial version of the Gotcha library, which will provide other tools (primarily performance analysis tools) with a better way to implement and control function wrapping.
- The AID project has had several publication on OpenMP race detection[89],[90] and floating point consistency[91].

Next Steps The ProTools team is starting a major new thrust in the Ubiquitous Performance Analysis effort. They are aiming to add Caliper into a targeted ASC code, and then build and integrated performance analysis workflow that brings together caliper and web-based performance visualization. The end result of this is to have application users running codes, producing behind-the-scenes performance data, and then application developers browsing and analyzing the performance data with analytic frameworks and novel visualizations.

The AID team is focusing on vendor interactions and co-design, for both CORAL systems and subsequent next-generation systems.

4.2.4 SNL ATDM Tools

Overview The SNL ATDM Tools project is broken into two subprojects: the SNL ATDM DevOps subproject, and the Sandia ATDM Performance Analysis subproject.

The SNL ATDM DevOps subproject focus is on tools and processes supporting DevOps (Development Operations) for the ATDM software development efforts. DevOps in the SNL ATDM context is all of the software infrastructure development, testing support, integration, and deployment work in support of the ATDM software application and component development teams. The primary activities of this subproject are to (1) coordinate and prioritize tasks for the various teams that provide DevOps support for ATDM codes, applications, and customers, (2) develop and help deploy shared build, test, and install infrastructure across the ATDM codes and projects, (3) define and support development, testing, integration, and other related workflows for ATDM projects.

The SNL ATDM Performance Analysis subproject is scoped with providing a broad cross-section of performance-related support activities for the laboratories ATDM efforts. These activities include: (1) providing support for high-performance, hardware-optimized cross-platform builds, including the generation of correct hardware compiler options/software defines; (2) performance analysis of benchmarking runs, including thread and node scaling, on relevant ASC testbeds and platforms, and (3) provision for algorithm/code modification or editing of run scripts to optimize performance where issues are identified.

The SNL ATDM Performance Analysis subproject also develops profiling and correctness tools which work with the Kokkos Profiling hooks API. These tools have been developed to provide insight into the timing of kernels written using Kokkos, as well as data structures utilizing Kokkos parallel containers or Views. In a number of cases, the profiling tools act as *connectors*, establishing a link between important Kokkos performance events and vendor provided tools such as Intel's VTune, NVIDIA's NSight and Arm's MAP profilers.

Key Challenges The key challenges associated with this project are the extremely aggressive porting and optimization requirements associated with Sandia's ATDM efforts. These activities are attempting to port and help support a minimum of three production applications, as well as multiple mini-applications and research prototypes to several ASC-relevant platforms. The first-of-a-kind algorithms being used on these platforms produce complex interactions in the applications that must be fully studied and analyzed to ensure a high level of performance is being offered to the Sandia's user base.

Combined with the application development effort, Sandia is investing heavily in the development of the Trilinos scalable solver stack (used by several codes in ECP and the broader HPC community). The Performance Analysis activity within ATDM is also providing low-level kernel and runtime optimization insight to developers in the Kokkos and Trilinos projects. The DevOps activity within SNL ATDM is providing configuration, build, testing, and workflows tools and processes to keep this stack of software working on the variety of platforms and configurations.

Solution Strategy The SNL ATDM Tools has the following primary thrusts:

1. **Common Build, Test, and Integration Tools** ensure scalable DevOps efforts and support.
2. **Testing and Integration Workflows** ensure smooth and productive development and deployment efforts for ATDM software on target platforms.
3. **High-Performance Applications** ensure well optimized application, library and kernel performance across ASC-relevant computing architectures.
4. **Performance Portability** ensures performance portability of Sandia ATDM codes across diverse ASC-relevant computing architectures.
5. **Lightweight Performance Tool Infrastructure** ensures that lightweight tools exist for rapid performance analysis or performance issue identification.

The Sandia Performance Analysis sub project was formed from the older Performance Modeling and Analysis Team in 2015. It's scope was refined to focus specifically on supporting application development activities

at the laboratories, with the intent to help provide much stronger levels of performance across the Sandia software portfolio. The project has provided significant application support since 2015 on topics including application porting and scaling on the ASC Trinity platform, porting to the ASC CTS-1 commodity clusters and has most recently been providing support for the forthcoming ASC Sierra platform housed at LLNL.

The Kokkos Profiling tools collection was formed in 2015 resulting from research efforts in several successful LDRD projects. The experimental interface to Kokkos was prototyped in 2014/5 and has since been the default configuration when compiling the Kokkos library.

Recent Progress For FY17, the SNL ATDM DevOps subproject completed a number of results: (1) created a TriBITS prototype build and test system for SPARC; (2) set up ATDM project and issue tracking utilizing JIRA and JIRA Portfolio; (3) worked to stabilize Trilinos for ATDM customers, and (4) worked with contractor Kitware to improve CMake/CTest/CDash and adding Fortran support in Ninja, performance, enhanced CDash. The Sandia Performance Analysis subproject provided the Sandia SPARC and EMPIRE applications with performance results on: (1) Intel Knights Landing many-core processors; (2) Intel Haswell multi-core processors; (3) IBM POWER8/NVIDIA P100 CORAL development systems; (4) ASC CTS-1 Broadwell processors, and, (5) early access ARM processors. The results of benchmarking and cross-platform kernel benchmark times were reported to developers with ATDM including the project leads for SPARC and EMPIRE as well as the Trilinos solver project.

For the first half of FY18, the DevOps subproject (1) developed integration workflows with Trilinos for the ATDM SPARC and EMPIRE Apps to shield them from instability in Trilinos and yet still drive co-development with Trilinos; and (2) set up initial ATDM builds of Trilinos for EMPIRE configuration submitting to CDash and addressed native Trilinos test suite failures. The Performance Analysis team has provided benchmark kernel timings for some important classes of kernels on: (1) Intel Skylake multi-core processors; (2) early-access IBM POWER9/NVIDIA Volta platforms (for CORAL activities), and, (3) additional ARM processors. The compilation flags and environment configurations have been integrated into Kokkos and Trilinos for wider community use.

Next Steps Our next efforts are:

1. **Complete upgrade of CMake/CTest/CDash**: Upgraded CMake/CTest/CDash will provide for faster builds and tests, better display and better query capability on CDash.
2. **Complete ATDM Trilinos builds**: Setup of testing on rest of platforms used by EMPIRE and then extend the ATDM Trilinos build configuration for SPARC.
3. **Transition ATDM APPs to use ATDM Trilinos builds:** SPARC and EMPIRE builds will use the same standard ATDM Trilinos build reducing duplicate work maintaining these builds, less computing resources to run the build, etc.
4. **Detailed Build Timing:** during FY18 the Performance Analysis subproject is investigating the timing of complex application builds including the time spent in file input/output, compilation itself, directory traversal and other metrics. The intention is to identify areas of optimization that will improve developer productivity (by reducing wait for builds to complete).
5. **Additional Benchmarking:** additional platforms and more extensive benchmarking activities are currently underway, particularly on CORAL POWER9 development systems. These studies will have improve the "day-one" performance of Sandia's application portfolio on the pre-Exascale Sierra platform when it is released to users during 2018.

4.2.5 Exascale Code Generation Toolkit

Overview Our project addresses the development of HPC software for exascale and similarly complex architectures. Generated code may be modified, wrapped in application specific APIs, or be checked into the user's application repository. Inputs to the code generation tools we have developed are either application code or simplified application code that represents naive implementations of algorithms. Input languages we are supporting include both legacy and modern versions of Fortran, C, and C++ (e.g., F77, F2008, F2015, C89, C99, C11, C++98, C++11, C++14, C++17, etc.). The output is generated code in the same language, but modified to address essential hardware-specific architectural features. We are supporting common language dialects including: UPC, OpenMP (for both C, C++, and Fortran), CUDA, and OpenCL.

Key Challenges Our project addresses the development of HPC software for exascale and similarly complex architectures. The approach in our project supports only the generation of code to simplify the job of the developers, and as such users are given complete control over how to use our code generation tools. However, this makes our project especially difficult because we must read and rewrite the user's source code application. This step requires significant compiler technology: analysis and transformations at the source code level so that it can leverage the vendor's compiler.

We operate using a source-to-source approach so that the target architecture's compiler (vendor compiler) may be used to full advantage. It is expected that the vendor's compiler will be best optimized (low level optimizations) to the vendor's architecture. However, the structure of code written (at a high level) to take advantage of the vendor compiler's optimization can be exceedingly complex, vary widely between vendors, and is thus well better suited to automated code generation.

Solution Strategy Figure 29 illustrates the approach used to process application code and automate transformations to support either performance optimization or correctness checking. Our project is developing

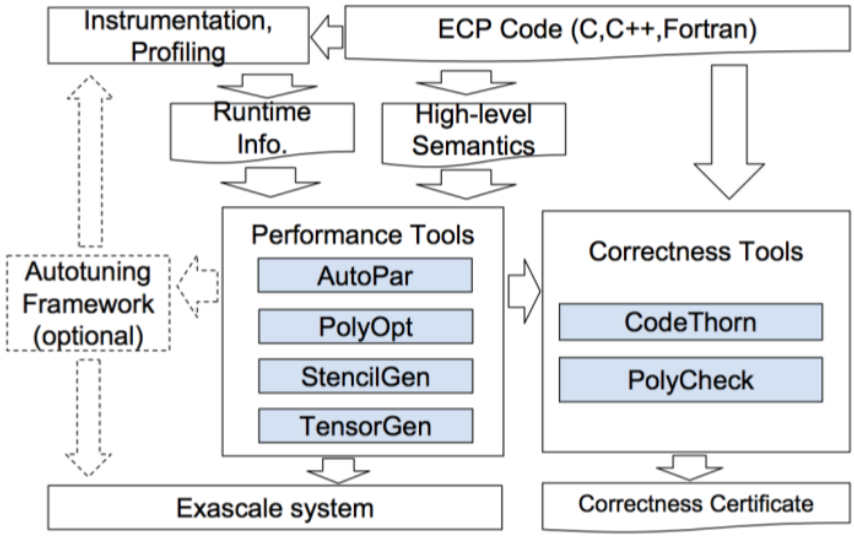


Figure 29: Approach to processing user application code with multiple tools to support optimization and correctness checking.

several tools for generating code: PolyOpt (a polyhedral optimizations program transformation engine capable of fully automating highly complex loop transformations), AutoPar (an automatic parallelization tool that inserts OpenMP directives into serial codes), CodeThorn (an award winning code correctness tool built using ROSE and SPOT), PolyCheck (a code correctness tool specific to polyhedral optimizations), and ROSE (a widely used compiler infrastructure for building specialized compiler tools). Our tools allow users to easily specify portions of an input application and automatically generate semantically equivalent, but higher performing code variants; including the ability to generate a multiple variants that traverse

a space of optimizations (e.g., loop tiling sizes, loop fusion, loop fission alternatives, etc.). The PolyOpt polyhedral optimizer is being enhanced with pattern-specific optimization and code generation strategies to address important patterns found in ECP codes. Stand-alone code generators developed previously by us for stencil computations and tensor contractions are being integrated with PolyOpt and their capabilities enhanced and hardened (StencilGen and TensorGen). The use of CodeThorn and PolyCheck have already been demonstrated for the verification of correctness of such automatically generated code.

Recent Progress We have made a release of TensorGen to the NWChem team and it has been used in their NWChem production release this past Fall (Fall 2017).

The StencilGen code generator takes as input a DSL specification of stencil functions and their domains, and creates CUDA code from it. The tool employs fusion, streaming, and overlapped tiling to achieve high performance. In order to avoid register spills for complex stencils, the tool also performs statement reordering that takes as input straight-line CUDA code for a multi-statement stencil and models it as a DAG of expression trees. The statements are then reordered to minimize register pressure [92].

The AutoPar tool accepts C/C++ serial programs as input and automatically generates OpenMP loop directives. We have continued to work on the CPU cost model to guide AutoPar's automatic parallelization. We have created a model based on the roofline model, and an existing tool in ROSE (the Arithmetic intensity tool) has been improved to provide key information for our model. We also added one option to use inlining in order to support loops with simple C function calls. Several microbenchmarks, including EPCC and STREAM, have been investigated and used to extract hardware and software metrics needed in our model.

CodeThorn takes as input polyhedral parallelized loops generated by the tool PolyOpt and checks whether the optimized loops are equivalent to the original (non-optimized) loops. We analyzed the availability of optimizable code patterns for verification in three proxy apps (AMG2013, CoMD, LULESH) and found 282 optimizable patterns that can serve as input to our verification. We extended the scope of the covered C++ subset for verification to support changes in the PolyOpt generated optimized code. We also extended the representation of program states to take pointers to dynamic data structures into account.

We have been developing the other tools mentioned above, adding supporting features, and testing to them. We gave a demo of these tools at the recent ECP all-hands meeting in February, 2018; and we distribute the working versions of these tools that we demonstrated at the meeting. These tools are regularly updated with ongoing work and released as part of Continuous Integration (CI) processes. Most tools are released as part of the ROSE distribution to simplify the testing and release process.

We have been adding new Fortran support to ROSE as part of our ECP Fortran support. We have also started the C++17 support in ROSE. We have compiled most of the ECP proxy applications as part of initial first year work. We now test the ROSE C and C++ compiler infrastructure against commercial compiler tests suits and it performs similarly to commercial compilers in initial testing for analysis tools.

Next Steps For StencilGen, we will develop several heuristics for fusion, streaming, and tiling based on architectural characteristics, as well as machinery to systematically explore various compositions of optimizations and optimization parameters. We will continue to work with the NWChem team at PNNL to support their computational code generation requirements. For CodeThorn we will (i) extend the evaluation with proxy apps and (ii) improve the reporting of detected semantic differences in the PolyOpt optimized code in comparison to the original code.

We are also working with the AMReX ECP team to support analysis and transformation of their application codes using our tools and future versions of them with additional features. The AMReX team is in turn also supporting multiple ECP application teams, to which all of our work is expected to apply directly.

4.2.6 Exa-PAPI

Overview The Exascale Performance Application Programming Interface (Exa-PAPI) project builds on the widely deployed and widely used Performance API (PAPI) and extends it with performance counter monitoring capabilities for new and advanced ECP hardware and software technologies, fine-grained power management support, and functionality for performance counter analysis at task granularity for task-based runtime systems. Exa-PAPI also adds events that originate from the ECP software stack (i.e., communication libraries, math libraries, task runtime systems, etc.) and, as a result, extends the notion of performance events from strictly hardware-related ones to include software-based information.

Exa-PAPI is essential for ECP because it enables the ECP application community to monitor both types of performance events—hardware- and software-related—in a uniform way, through one consistent PAPI interface. On the hardware side, Exa-PAPI provides access to a wide range of new events for the extreme-scale platforms that will form the basis of exascale computing. Furthermore, it provides a finer-grain measurement and control of power, thus offering software developers a basic building block for dynamic application optimization under power constraint. In addition to providing hardware counter based information, Exa-PAPI integrates a standardizing layer for monitoring software-defined events (SDEs), which will expose the internal behavior of runtime systems and libraries to the applications. Addressing the gap of software-defined event monitoring—and enabling monitoring of both types of performance events though Exa-PAPI—stands to offer a transformative impact on performance analysis and application development as a whole.

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 project prepares the PAPI library to stand up to the challenges posed by exascale systems by: (1) widening its applicability and providing robust support for hardware resources that are currently out of PAPI's scope; (2) supporting new programming paradigms, such as task-based systems, by adding functionality for performance counter analysis at task granularity (as opposed to core and thread granularity); (3) extending PAPI to support software-defined events, in addition to the traditional hardware-based 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.

The Exa-PAPI effort delivers new PAPI components to handle the wide range of new hardware and software events for the extreme scale platforms that will form the basis of exascale computing. To achieve this, Exa-PAPI implements a variety of monitoring and sampling capabilities for the different technologies, which are exported to the ECP application community. Exa-PAPI also provides finer-grain measurement and control of power, thus offering software developers a basic building block for dynamic application optimization under power constraint. Other hardware efforts in Exa-PAPI are the development of components for monitoring network interconnect events, as well as components targeted at the deep and heterogeneous memory hierarchies that we are already seeing in new architectures.

Recent Progress The Exa-PAPI hardware and power effort began with the implementation of new PAPI components enabling Intel Knights Landing (KNL) hardware counter and power management support. In December 2017, the latest version of PAPI (5.6.0) was shipped, releasing two new components that are fully integrated into the PAPI library for KNL core and uncore support. Additionally, PAPI ships with a powercap component for power/energy measurement and control. This development delivers two improvements. First, in the past, PAPI power components supported only *reading* power information. The new component exposes running average power limit (RAPL) functionality to allow users to read and write power. Second, the

original PAPI power component accessed the RAPL model-specific registers (MSRs) directly, and, therefore, reading power data required root privileges. The new PAPI power component uses the powercap interface that comes built-in with the Linux kernel. The purpose of this interface is to expose the RAPL settings to user-space. Therefore, power reading is possible without any superuser privileges—only Linux kernel version 3.13 (or higher) is required.

Since the concept of writing (or capping) power is new to PAPI, we studied numerical building blocks of varying computational intensity, and used the PAPI powercap component to detect power optimization opportunities. We experimented with a wide range of power caps on the KNL architecture to reduce the power usage for different numerical kernels while keeping the execution time constant so that real energy savings can be achieved. Figure 30 shows one example where we use the Jacobi iterative method to solve a finite difference discretization of the Helmholtz equation. While the default power consumption is around 185 Watts, with power capping, we were able to improve the energy efficiency by 25% without any loss in time-to-solution. All our findings have been published in a conference paper [93] and a journal paper [94].

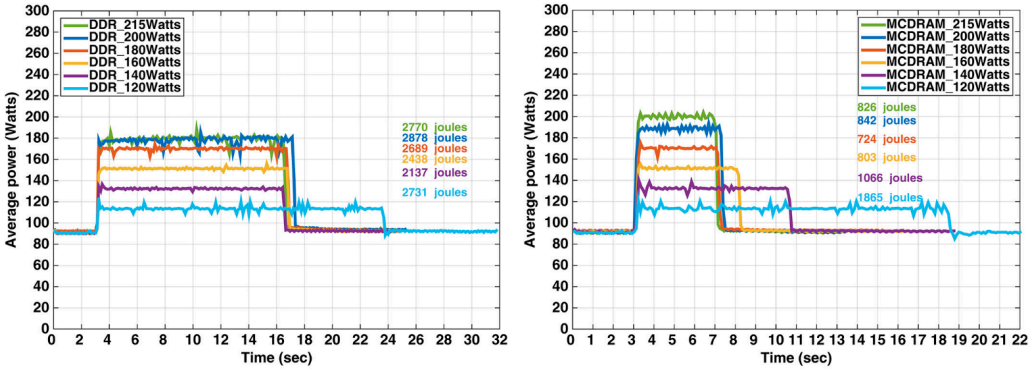


Figure 30: Average power measurements (Watts on y axis) of Jacobi algorithm on a 12,800 x 12,800 grid for different power caps. (A) FLAT mode: data allocated to DDR4; (B) FLAT mode: data allocated to MCDRAM

On the software-defined events front, we have already proposed an API (publicly available on Jira: https://jira.exascaleproject.org/secure/attachment/12251/2017_SDE_API_report.pdf) and received significant feedback and requests for changes by members of the runtime and library communities, which we have incorporated. We have also developed a prototype implementation of an SDE component in PAPI, which we are using to integrate SDE support in various ECP projects, such as:

1. ByFL (HT-DSE): https://bitbucket.org/jagode/byfl_papi_sde
2. PaRSEC (2.3.1.09 STPM11-ParSEC): https://bitbucket.org/herault/parsec/branch/PAPI-SDE
3. PEEKS (2.3.3.10 STMS11-PEEKS): https://bitbucket.org/icl/magma (branch: `PAPI_SDE`)
4. NWchemEx (2.2.1.02 ADSE11-NWChemEx): https://bitbucket.org/jagode/nwchem_papi_sde

Next Steps Our next efforts will focus on:

1. **Development of a PAPI component for IBM Power9 Hardware Counter Support:** Add support for (1) core performance events, which are specific to each core; and (2) shared events, which monitor the performance of node-wide resources that are shared between cores. Access to shared events require elevated privileges. However, IBM's official route for providing access to shared events will be through the Performance Co-Pilot (PCP) for non-root users. Thus, one of the Exa-PAPI efforts is to develop a PAPI-PCP component so that all users can access Power9 shared events through PAPI.

2. **Release of PAPI's SDE component, and integration of SDEs with other ECP efforts**: Refine the PAPI SDE prototype implementation based on feedback from the ECP community and experience acquired from instrumenting ECP projects. Instrument ECP libraries, runtimes, and applications, such as PaRSEC, PEEKS, and NWChem to use software-defined events to export performance information.

4.2.7 2.3.2.07-YTune

Overview We are developing tools and an application development workflow that separates a high-level C/C++/FORTRAN implementation from 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, 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.Thus, autotuning will mitigate the need for extensive manual tuning.

Autotuning is essential for ECP in 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. Therefore, if Exascale application developers are expressing their computation and separating that from its mapping to hardware, autotuning 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.

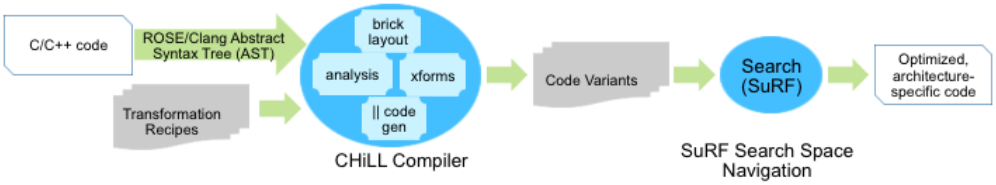


Figure 31: Y-TUNE Solution Approach.

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 to impact performance of their codes.

The autotuning compiler strategy revolves around the approach of the CHiLL autotuning compiler, 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.

We have developed a *brick data layout library and code generator* for stencil computations within CHiLL. Recent trends in computer architecture that favor computation over data movement incentivize high-order methods. Paradoxically, high-order codes can be challenging for compilers/optimization to attain

high performance. Bricks enable high performance and make fine-grained data reuse and memory access information known at compile time. The SIMD code generation achieves performance portability for high-order stencils for both CPUs with wide SIMD units (such as Intel Knights Landing) and GPUs. Integration with autotuning achieves performance that is close to roofline performance bounds for both architectures.

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

Recent Progress We have pursued the following main activities since the beginning of 2018:

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

SuRF for SuperLU and QMCPACK: We focused on testing and hardening SuRF for tuning SuperLU package. We used 6 matrices that come from different DOE applications and ran SuRF in an asynchronous mode with up to 32 nodes. We compared the results from SuRF to those from OpenTuner. On all instances tested, we found that SuRF obtains comparable results but in half the time of OpenTuner. We also observed that SuRF found high quality solutions in short computation time and used the remaining time for neighborhood exploration. Therefore, we implemented early stopping criterion. We also did single node tuning experiments with QMC. Since the current search space of QMCPACK is rather small, we did not evaluate it at scale. Currently, we are working with the QMCPACK developers to expose more parameters. Recently, we developed stopping criterion based on local convergence and expected improvement over time. This allows the search to terminate in shorter computation time. Currently, we are expanding the search for multinode autotuning where each evaluation spans multiple nodes.

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 P100 (Pascal) GPU.

Next Steps In the near future, we will release the CHiLL autotuning compiler, and migrate SW4 to use the brick data layout. We will continue the transition of CHiLL capabilities to LLVM. In SuRF, we plan to explore multinode search, and integrate SuRF into the compiler-directed autotuning we are doing.

4.2.8 HPCToolkit

Overview The HPCToolkit project is working to develop performance measurement and analysis tools to help ECP software developers understand where and why their programs do 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 applications as they execute on existing ECP testbeds and new technologies needed to measure and analyze performance on forthcoming Exascale systems.

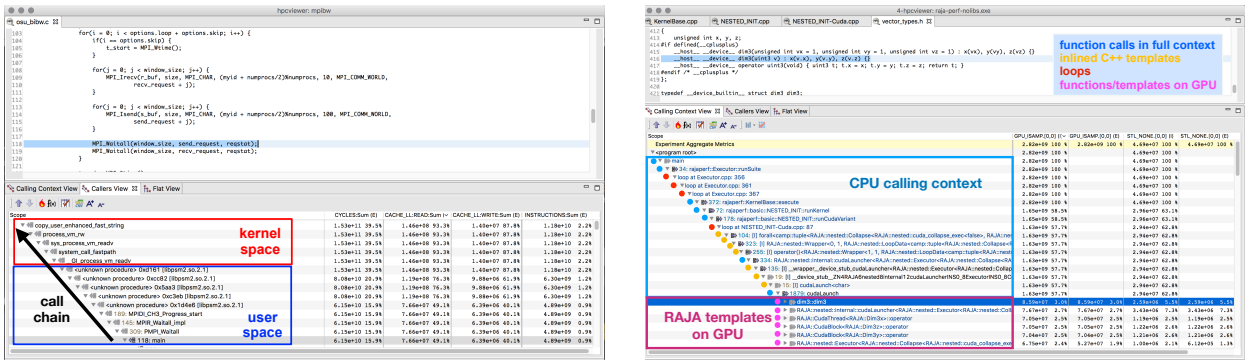
To provide a foundation for performance measurement and analysis, the project team is working with community stakeholders, including standards committees, vendors, and open source developers to improve hardware and software support for measurement and attribution of application performance on extreme-scale parallel systems. The project team has been engaging vendors to improve hardware support for performance measurement in next generation systems 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, the project team is working to extend capabilities to measure computation, data movement, communication, and I/O as a program executes to pinpoint scalability bottlenecks, evaluate resource consumption, and quantify inefficiencies.

Key Challenges In recent years, the complexity, diversity, and the rate of change of architectures for extreme-scale parallel systems have increased dramatically. For higher efficiency, heterogeneous designs that couple multicore processors with accelerators and employ more complex memory hierarchies have been increasing in importance. In addition, the DOE is purposefully pursuing multiple independent architectural designs for next generation parallel systems as part of risk mitigation. For performance tools, the need to support multiple diverse architectural paths 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 all of the idiosyncratic 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 more complex memory hierarchies and accelerators. Second, the team is working to improve performance attribution given optimized code for complex node-level programming models used by ECP developers, including OpenMP and template-based programming models such as LLNL's RAJA and Sandia's KOKKOS. To support this effort, the project team is enhancing the Dyninst binary analysis toolkit, which is also used by other ECP tools. 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. Fifth, the team will enhance HPCToolkit's user interfaces to help analyze performance bottlenecks on extreme-scale 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.

Recent Progress Over the last year, the HPCToolkit project has significantly enhanced the ability to measure and analyze application performance.

- The project team added a new measurement substrate to HPCToolkit to measure performance using the Linux perf_events interface. Using perf_events enables HPCToolkit to measure operating system



(a) HPCToolkit kernel activity performance metrics. **(b)** HPCToolkit GPU offloaded performance.

Figure 32: The December 2017 HPCToolkit release supports measuring and attributing performance metrics of kernel activity on behalf of an application. HPCToolkit now measures and attributes the performance of computation offloaded to GPUs using LLNL's RAJA template-based programming model.

activity and thread blocking in addition to application execution. Figure 32a displays a screenshot of HPCToolkit's code-centric user interface that shows how HPCToolkit reports information about kernel activity on behalf of an application as part of an application's performance.

- To accurately attribute code performance to elements of complex, parallel software frameworks that have been transformed by optimizing compilers, HPCToolkit employs an approach that combines information recorded by compilers about line maps and the provenance of inlined code with direct analysis of machine code to recover information about a program's control flow. Over the past year, the project team has developed improved techniques for recovering control flow graphs from machine code and employed them to relate application performance to inlined functions, templates, and loops in highly optimized code on both host processors and attached accelerators. Figure 32b shows the precise attribution of performance measurements to C++ templates employed as part of LLNL's RAJA portability layer.
- The project team has developed novel capabilities for measurement, analysis, and attribution of applications that employ graphics processing units (GPUs) as accelerators. This work includes leading the design of the OMPT tool application programming interface as part of the emerging OpenMP 5.0 standard, developing a measurement infrastructure as part of `libomptarget`—an open source library for offloading code onto accelerators, enhancing HPCToolkit to ingest measurement data from accelerators, and extending HPCToolkit to analyze binaries for NVIDIA's GPUs to attribute performance of offloaded code. Figure 32b illustrates how HPCToolkit can attribute the performance of code offloaded onto a GPU to the host context that offloaded the computation.

Next Steps The next steps in the project are to:

- Work with the OpenMP standards committee to finalize tool interfaces as part of the emerging OpenMP standard.
- Complete and deploy implementation of HPCToolkit's support for measurement and analysis of code offloaded onto NVIDIA GPUs.
- Integrate new support for task-based parallelism developed as part of the project's Dyninst binary analysis infrastructure into HPCToolkit's binary analyzer to accelerate analysis of large executables.
- Complete work on data-centric performance analysis capabilities that measure and attribute data movement costs to program variables.
- Complete and deploy a framework for regression testing of HPCToolkit.
- Work with DOE and Intel on performance measurement technologies for the A21 Exascale platform.

4.2.9 PROTEAS: Programming Toolchain for Emerging Architectures and Systems

Key Challenges: Programmer productivity and performance portability are two of the most important challenges facing applications targeting future Exascale computing platforms. Application developers targeting evolving ECP architectures will find it increasingly difficult to meet these dual challenges without help from integrated capabilities that allow for flexibility, composability, and interoperability across a mixture of programming, runtime, and architectural components. In particular, an integrated programming toolchain is critical for Exascale delivery. First, it will provide a programming pathway to anticipated Exascale architectures by addressing programmability and portability concerns of emerging technology trends seen in pre-procurement machines. It will also enable ECP applications teams to explore programming options to find the most effective and productive approaches without constraining programming models or software solutions. Second, an integrated programming framework strategy will deliver solutions that will be further refined for the architecture capabilities known to be in the system procurement. This is essential for maintaining developer productivity and attaining performance portability as ECP requirements evolve.

Solution Strategy: The PROTEAS (PROgramming Toolchain for Emerging Architectures and Systems) project is a strategic response to the continuous changes in architectures and hardware that are defining the landscape for emerging ECP systems. PROTEAS is 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 objectives are to

1. Provide productive and performance-portable programming solutions based on directive-based methodologies that support current language paradigms and flexible prototyping of interfaces specifically directed at heterogeneous and manycore processors, deep memory hierarchies, and nonvolatile memory systems (NVM);
2. Provide integrated performance assessment solutions for these programming systems that will enable automatic performance analysis and performance-driven optimization;
3. Provide an integrated programming toolchain that is powerful enough to prototype the above solutions, while flexible enough to extend its functionality over time;
4. Refine our toolchain and solutions through engagement with ECP applications teams who will evaluate prototypes, provide feedback, promote application readiness, and facilitate use of ECP prototype and eventual production machines; and,
5. Champion our successful solutions in ECP procurements, community standards, and open-source software stacks.

Our team has started with a strong existing base of relevant technological and software capabilities. Importantly, our solutions are based on our significant, continuing work with LLVM, ARES HLIR, OpenARC, and TAU. We have 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.

Our strong emphasis on delivering an effective toolchain to application developers within the next few years emphasizes the importance of adopting an integrated programming solution that will be further refined for the architecture capabilities known to be in the Exascale system procurement. We will develop an integrated system (i.e. compilers, runtime systems, debuggers, and performance tools) suitable for deployment in the 2019 timeframe. The experience gained from this development will inform vendor collaborations, proposals to standards committees, and existing open source software to make key elements of our developed technology commercially available for ECP deployment in the 2021 timeframe.

While PROTEAS will be oriented towards foreseeable architectural trends, it will not lock in to specific choices that will constrain what new hardware features it can address. Rather, it is important for the programming framework to embody interoperability, open interfaces, and flexibility in the toolchain, allowing it to pursue high-value solutions as opportunities arise and thereby achieve Exascale performance potential.

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

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

2. Papyrus for portability across NVM architectures. Develop a portable interface to NVM architectures to provide massive, persistent data structures as required by many applications. See §4.2.11.

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 §4.2.10.

4. Improving LLVM. In collaboration with numerous other ECP projects, PROTEAS is contributing improvements to the LLVM compiler infrastructure. These improvements include simple bugfixes to the existing infrastructure, monitoring Flang progress, developing Clacc (see §4.2.12), and contributing to the development of a new parallel intermediate representation (see `https://github.com/Parallel-IR/llvm-pir/wiki`).

5. Outreach and collaboration with ECP applications teams. We have interacted with over a dozen applications teams to help prepare their applications for ECP. See §4.2.12, §4.2.11, and §4.2.10.

Next Steps: Our next efforts are:

1. Clacc. Complete Clacc support for a prescriptive interpretation of OpenACC by lowering OpenACC directives to use the existing LLVM OpenMP infrastructure.

2. Papyrus. Improve support for data compress and encryption to provide enhanced storage reduction and performance improvements, and security for sensitive data, respectively.

3. Tau. Improve performance instrumentation for deep memory hierarchies in Tau, focusing primarily on KNL MCDRAM and NVM.

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

4.2.10 PROTEAS — TAU Performance System

Overview The TAU Performance System is a versatile profiling and tracing toolkit that supports performance instrumentation, measurement, and analysis. Figure 33 gives an example of using TAU's parallel profile analysis tool, ParaProf. 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, and CUDA in TAU.

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

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

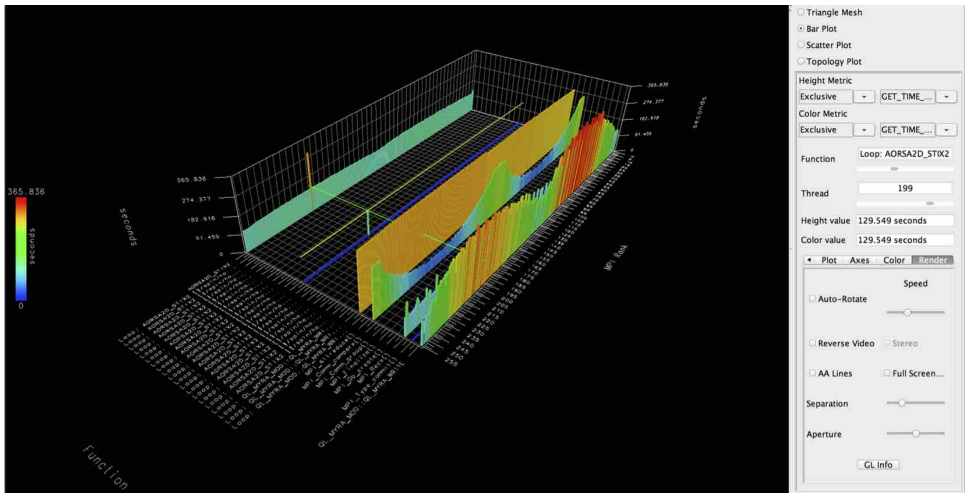


Figure 33: TAU's ParaProf profile browser shows the parallel performance of the AORSA2D application.

Recent Progress

1. **Compiler-based instrumentation:** Added support for compiler-based instrumentation in TAU for LLVM Clang and Flang compilers on IBM Power 9, Intel x86_64, and Cray XC systems.
2. **Kokkos** Added support for the Kokkos profiling interface in TAU by extending the tau_exec tool that preloads the TAU DSO in an uninstrumented binary and applied it to ECP benchmarks.
3. **CANDLE** Extended TAU to support performance evaluation of Python and CUDA and applied it to evaluate the performance of the CANDLE ECP Benchmarks on IBM Power and Cray XC systems.
4. **Improved CUDA and OpenMP support** Added support for newer GPUs and enhancements to the CUPTI profiling interface in TAU. Updated the OpenMP Tools Interface support in TAU.

Next Steps

1. **NVM instrumentation** Design and implement support for supporting deep memory hierarchies in TAU for supporting MCDRAM based systems. This includes support for memkind hbm-malloc and Fortran FASTMEM directives.
2. **PHIRE** Design and implement support for instrumentation of higher level PHIRE intermediate language constructs in TAU.
3. **Outreach** Outreach activities to demonstrate comprehensive performance evaluation support in TAU for OpenARC, LLVM, CUDA, Kokkos, and NVM based programming frameworks for SHC platforms.

4.2.11 PROTEAS — 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) [95], C++ template container library (TCL) [95], and key-value store (KV) [96]. (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, recently-announced 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 Meraculous [97] is a state-of-the-art de novo assembler written in UPC. Its parallel algorithm for de Bruijn graph construction and traversal leverages the one-sided communication in UPC to facilitate the requisite random access pattern in the global de Bruijn graph. The de Bruijn graph is implemented as a distributed hash table with an overlapping substring of length k, referred to as a *k-mer*, as key and a two-letter code [ACGT][ACGT] as value as shown in Figure 34. A hash function is used to define the affinities between UPC threads and hash table entries. We ported the distributed hash table written in UPC to a PapyrusKV database. The keys in the database are k-mers and the values are two-letter codes. The PapyrusKV runtime calls the same hash function in the UPC application to determine the owners of key-value pairs in the database by specifying the custom hash function when the database is created. Thus, the thread-data affinities in UPC and PapyrusKV are the same as shown in Figure 34. PapyrusKV requires fewer lines of source code than UPC because it calls standard put and get API functions without implementing an application-specific algorithm for the distributed hash table construction and traversal. Figure 35 shows the performance comparison between PapyrusKV and UPC of Meraculous on Cori. Both versions are built and run using Berkeley UPC, an MPI-interoperable UPC implementation. We measured the total execution time on 32, 64, 128, 256, and 512 UPC threads (32 UPC threads per node). UPC shows better performance than PapyrusKV due to its RDMA capability and built-in remote atomic operations during the graph traversal. The performance gap between UPC and PapyrusKV decreases as the number of UPC threads increases. On 512 UPC threads, PapyrusKV runs 1.5 times slower than UPC. This is mainly because of the asynchronous migration in PapyrusKV during the graph construction.

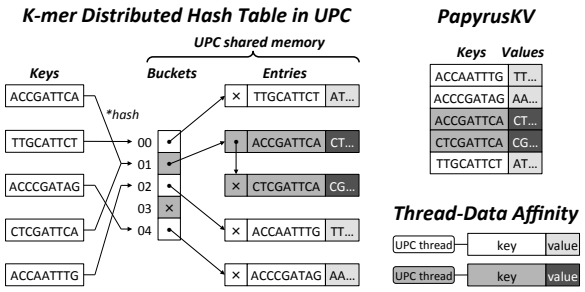


Figure 34: K-mer distributed hash table implementations in UPC and PapyrusKV.

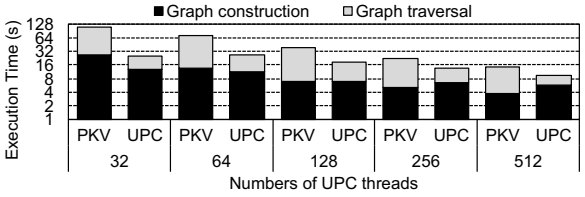


Figure 35: Meraculous performance comparison between PapyrusKV (PKV) and UPC on Cori.

Next Steps Our next efforts are:

1. **Data compression:** The overhead of data access and movement becomes a serious bottleneck compared to compute overhead in large-scale HPC systems. We will integrate data compression techniques into Papyrus to achieve storage reduction and performance improvement.

2. **Data encryption:** More sensitive data (e.g., health records, DNA data) is being used in distributed infrastructures, and users need practical methods to secure their data throughout its lifecycle. We will introduce data encryption in Papyrus to add an extra layer of security in the complex scientific workflows.

4.2.12 PROTEAS — 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 terms of functionality, performance, programmability, and scalability, significantly increasing the complexity that ECP application developers face as they attempt to fully utilize the available hardware.

A key enabling technology being pursued as part of the PROTEAS project is OpenACC. While OpenMP has historically focused on shared-memory multi-core programming, 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 [98]. However, GCC has lagged behind commercial compilers, such as PGI's, in providing production-quality support for the latest OpenACC specifications [99]. 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.

A major goal of PROTEAS 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. We call this project *clacc*. The PROTEAS objectives for clacc are:

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

2. As part of the compiler 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. As the work matures, contribute OpenACC support to upstream clang/LLVM so that it can be used by the broader HPC and parallel programming communities.

4. Throughout development, actively contribute upstream any clang/LLVM improvements that are mutually beneficial to both our OpenACC work and to the broader clang/LLVM ecosystem.

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.

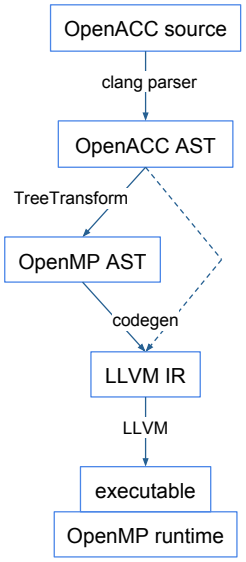
2. **Source-to-Source:** Source-to-source translation from OpenACC to another programming language can significantly reduce the effort to implement OpenACC. However, a well known issue with LLVM's compiler front end, clang, is that its AST, the source-level representation, was designed to be immutable. Moreover, analysis and optimization capabilities are implemented at the level of the LLVM intermediate representation (IR) not at the AST level, but such capabilities would be critical for lowering OpenACC's descriptive language to a more prescriptive language, like OpenMP.

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 feature of the clacc design is to lower OpenACC to OpenMP. This design has several benefits:
   (a) By building on clang/LLVM's existing OpenMP compiler and runtime support, it reduces the effort necessary to construct a production-quality OpenACC implementation.
   (b) It facilitates repurposing for OpenACC existing OpenMP static analysis and debugging tools.
   (c) It facilitates porting applications from OpenACC to OpenMP to alleviate the aforementioned concerns about developing applications in OpenACC.
2. To ensure clacc's successful implementation and eventual acceptance upstream, we have begun and will continue design discussions with the clang/LLVM communities throughout clacc's development.
3. To handle clang's immutable AST, clacc's design reuses a clang feature called TreeTransform, which was originally designed for C++ template specializations.
4. To take advantage of analyses and optimizations at the LLVM IR level, we are investigating ongoing efforts to develop a parallel LLVM IR, which clacc could use as an alternative code generation target.
5. To stage our development effort, we are initially implementing clacc with two simplifications: we are implementing a prescriptive interpretation of OpenACC to achieve correct behavior, and we are implementing and testing only within C. We will extend this implementation with the necessary analyses and optimizations for a descriptive interpretation and for C++ afterward.
6. Throughout clacc development, we are continuously integrating the latest upstream clang/LLVM changes, and we are running and extending the clang/LLVM test suites to detect regressions and incompatibilities. We are also investigating OpenACC benchmarks [100] and validation test suites [99] to ensure correct OpenACC behavior and good performance.

Recent Progress

1. Prototyped the translation of an initial set of OpenACC directives and clauses to OpenMP.
2. Investigated OpenACC applications, benchmarks, and validation test suites for use in clacc testing. Reached out to ECP application teams who have expressed interest in OpenACC.
3. Initiated clacc design discussions within the clang/LLVM developer community.
4. Contributed to upstream clang/LLVM a number of fixes and other improvements to clang attribute and printing support, the clang/LLVM testing infrastructure, and the OpenMP implementation.

Next Steps

1. Complete clacc support for a prescriptive interpretation of OpenACC for correct behavior, and continue to contribute mutually beneficial improvements to upstream clang/LLVM as we develop them.
2. Continue clacc design discussions with the clang/LLVM developer community.
3. Explore applications from ECP teams we have previously contacted.

4.3 MATHEMATICAL LIBRARIES

This section present projects in Mathematical Libraries.

4.3.1 xSDK4ECP

Overview The xSDK4ECP project is creating a value-added aggregation of DOE math and scientific libraries through the *xSDK* (Extreme-scale Scientific Software Development Kit) [101], 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 use diverse, independently developed xSDK libraries in combination to solve large-scale multiphysics and multiscale problems. We are extending draft xSDK package community policies and developing interoperability layers among numerical libraries in order to improve code quality, access, usability, interoperability, and sustainability. Focus areas are (1) coordinated use of on-node resources, (2) integrated execution (control inversion and adaptive execution strategies), and (3) coordinated and sustainable documentation, testing, packaging, and deployment.

xSDK4ECP is needed for ECP because it enables ECP apps such as ExaAM and ExaWind to seamlessly leverage the entire scientific libraries ecosystem. For example, ExaWind has extremely challenging linear solver scaling problems. xSDK4ECP provides access to all scalable linear solvers with minimal changes. xSDK4ECP is also an essential element of the product release process for ECP ST. xSDK4ECP provides an aggregate build and install capability for all ECP math libraries that supports hierarchical, modular installation of ECP software. Finally, xSDK4ECP provides a forum for collaborative math library development, helping independent teams to accelerate adoption of best practices, enabling interoperability of independently developed libraries and improving developer productivity and sustainability of the ECP ST software product.

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

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

Solution Strategy The xSDK effort has two primary thrusts:

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

xSDK interoperability efforts began first with eliminating incompatibilities that prohibited correct compilation and integration of the independently developed libraries. These issues include being able to use a common version of a library such as SuperLU by PETSc and Trilinos. The second, and ongoing phase is increased use of one package's capabilities from another. For example, users who build data objects using PETSc can now access Trilinos solvers without copying to Trilinos data structures.

xSDK community package policies [102, 103] are a set of minimum requirements (including topics of configuring, installing, testing, MPI usage, portability, contact and version information, open source licensing, namespacing, and repository access) that a software package must satisfy in order to be considered xSDK compatible. The designation of xSDK compatibility informs potential users that a package can be easily used with others.

xSDK community installation policies [104] help make configuration and installation of xSDK software and other HPC packages as efficient as possible on common platforms, including standard Linux distributions and Mac OS X, as well as on target machines currently available at DOE computing facilities (ALCF, NERSC, and OLCF) and eventually on new Exascale platforms.

Community policies for the xSDK promote long-term sustainability and interoperability among packages, as a foundation for supporting complex multiphysics and multiscale ECP applications. In addition, because

new xSDK packages will follow the same standard, installation software and package managers (for example, Spack [4]) can easily be extended to install many packages automatically.

Recent Progress Figure 36 illustrates a new *Multiphysics Application C*, built from two complementary applications that can readily employ any libraries in the xSDK, shown in green. Current xSDK member packages (version 0.3.0, released December 2017) are the four founding libraries (hypre [105], PETSc [106], SuperLU [107], and Trilinos [108]) and three additional ECP math libraries added during this release (MAGMA [109], MFEM [9], and SUNDIALS [110]). Application domain components are represented in orange. Of particular note is Alquimia [111], a domain-specific interface that support uniform access to multiple biogeochemistry capabilities, including PFLOTRAN [112]. Additional ECP math libraries are working toward becoming xSDK member packages and plan to participate in future xSDK releases.

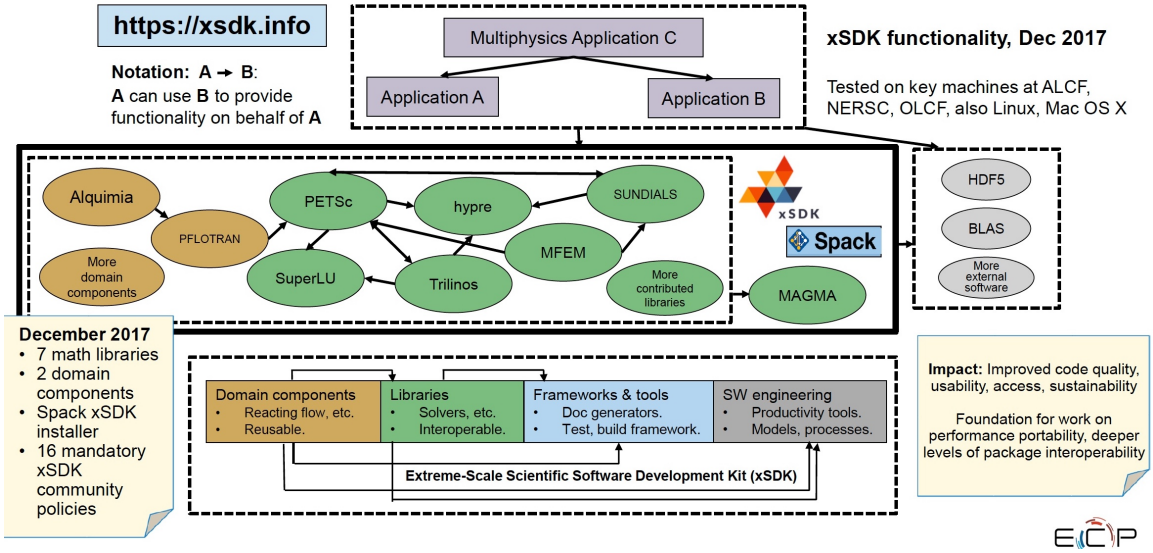


Figure 36: The December 2017 release of the xSDK contains many of the most popular math and scientific libraries used in HPC. The above diagram shows the interoperability of the libraries and a multiphysics or multiscale application.

The arrows among the xSDK libraries indicate current support for a package to call another to provide scalable linear solvers functionality on its behalf. For example, *Application A* could use PETSc for an implicit-explicit time advance, which in turn could interface to SuperLU to solve the resulting linear systems with a sparse direct solver. *Application B* could use Trilinos to solve a nonlinear system, which in turn could interface to hypre to solve the resulting linear systems with algebraic multigrid. Of course, many other combinations of solver interoperability are also possible. The website `https://xsdk.info/example-usage` and [113] provide examples of xSDK usage, including interoperability among linear solvers in hypre, PETSc, SuperLU, and Trilinos.

Next Steps Our next efforts are:

1. **Include more libraries:** xSDK4ECP will continue efforts to expand the number of participating packages, adapt community policies, and exploit increased interoperability. The next phase of xSDK packages will include deal.II, a popular finite element library that has not been funded by DOE. We anticipate some adaptation of language and policies that may be DOE centric.

2. **Process control transfer interfaces:** The ever-increasing use of concurrency within the top-level MPI processes requires that computational resources used by an application or library can be transferred to another library. Transfer of these resources is essential for obtaining good performance. The xSDK project will develop interfaces to support sharing and transfer of computational resources.

4.3.2 hypre

Overview The *hypre* software library [8, 114] provides high performance preconditioners and solvers for the solution of large sparse linear systems on massively parallel computers, with particular focus on algebraic multigrid solvers. One of *hypre*'s unique features is the provision of a (semi)-structured interface, in addition to a traditional linear-algebra based interface. The semi-structured interface is appropriate for applications whose grids are mostly structured, but with some unstructured features. Examples include block-structured grids, composite grids in structured adaptive mesh refinement (AMR) applications, and overset grids. These interfaces give application users a more natural means for describing their linear systems, and provide access to methods such as structured multigrid solvers, which can take advantage of the additional information beyond just the matrix. Since current architecture trends are favoring regular compute patterns to achieve high performance, the ability to express structure has become much more important. The *hypre* library provides both unstructured and structured multigrid solvers, which have shown excellent scalability on a variety of high performance computers, e.g Blue Gene systems (unstructured solver BoomerAMG has scaled up to 1.25 million MPI cores with a total of 4.5 million hardware threads). It is used by many ECP application teams, including ExaAM, Subsurface, ExaWind, CEED, and more. It requires a C compiler and an MPI implementation, but it also runs in an OpenMP environment. It has some GPU capabilities.

Key Challenges While *hypre*'s solvers contain much parallelism, their main focus is the solution of sparse linear systems, leading to very large demands on memory bandwidth. In addition, the use of multiple levels, while greatly aiding convergence of the solvers, leads to decreasing systems sizes, number of operations and parallel efficiencies on coarser levels. Particularly the unstructured algebraic multigrid solver BoomerAMG[115], 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. There are several algorithms that are more suitable for GPUs, such as direct interpolation, which however leads to degraded convergence. It could possibly be improved using Jacobi interpolation. All these options would need to be implemented and tested on GPUs. 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 the upcoming computer architectures are heterogeneous with accelerators, it was very important to enable *hypre* for GPUs. We looked into various options, such as the use of CUDA, OpenMP 4.5, as well as RAJA and Kokkos. We limited the latter two options to the structured interface and solvers which are more natural candidates for such an approach due to their use of macros, called BoxLoops, for loops. Since computer architectures continue to change rapidly, it is important to come up with strategies that will facilitate future porting of the software. Therefore we decided to develop a new memory model that addresses the use of different memory locations.

Recent Progress Under internal LLNL funding we pursued the following ECP-related tasks: enabling portions of several solvers for GPUs, and introducing a new memory model that is based on an abstract machine model.

We implemented the new memory model. The new model modified *hypre*'s memory allocation and copying routines to include memory allocations: HYPRE_MEMORY_HOST, HYPRE_MEMORY_DEVICE and HYPRE_MEMORY_SHARED. It also includes a new routine SetExecutionMode that can be used to define where the current code can be run. The new model can be mapped to the actual hardware in a variety of ways through the configure process, e.g., a host-only machine (all allocations are just mallocs) or with or without unified memory. Since it is based on an abstract machine model, it is expected that it will increase

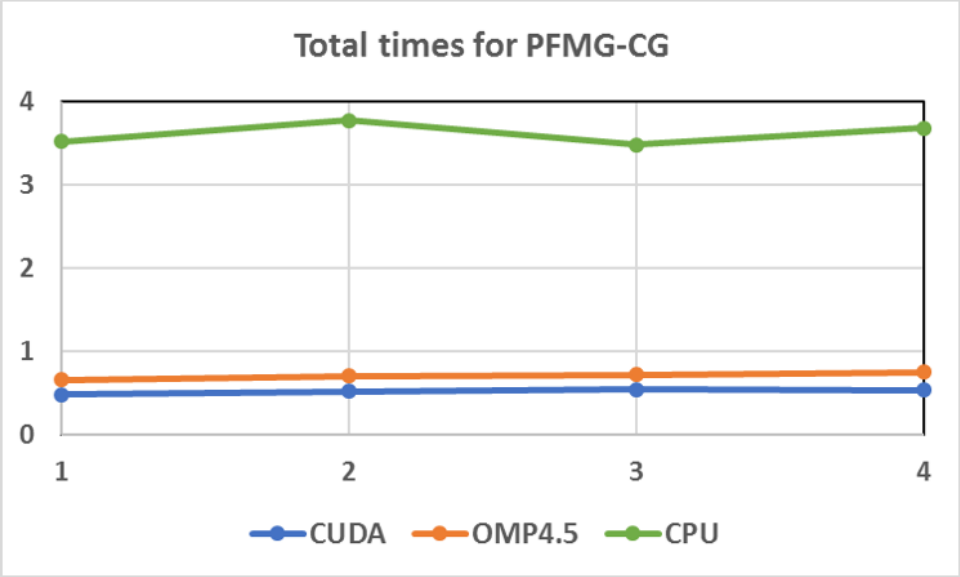


Figure 37: Performance of PFMG-PCG on Ray at LLNL, using Power 8 CPUs and Pascal GPUs

portability to future architectures. The implementation of the model was very intensive, since it affected the whole library.

In addition, we implemented various GPU capabilities in *hypre*. For the structured solvers, SMG and PFMG[116], both setup and solve phase can now completely be run on GPUs, using both CUDA or OpenMP4.5, and do not require unified memory. In addition, options to use RAJA and Kokkos are available, albeit not well tested yet. Figure 37 shows the performance of the total run times, including both setup and solve phase, for our fastest performing multigrid solver PFMG-CG for a 7-point 3D Laplace problem using 32 million grid points per node, comparing performance on the CPU only utilizing all its cores, to running the same problem on the GPUs using CUDA and OpenMP 4.5.

Porting the unstructured solver, BoomerAMG turned out to be far more complex. Currently only the solve phase can be run on the GPU for select smoothers, mainly Jacobi smoothers, and requires unified memory. The setup phase can currently be performed on the CPU only.

Next Steps Our most immediate plans are to improve the efficiency of interfacing applications with *hypre*'s solvers. This includes removal of unnecessary copies, and increasing use of threading, wherever possible. We plan to remove the requirement to use unified memory for BoomerAMG and want to increase the portions of its solve phase that are not GPU-enabled yet, but are well suited for GPUs, e.g. polynomial smoothers. Further out we would like to investigate how to enable the setup phase for GPUs, initially porting algorithms that are suitable as well as finding ways to improve convergence where necessary. In addition, we would like to work with ECP application teams who are using *hypre* or would like to use it, to achieve best performance by tuning the solvers for them and potentially implementing suitable algorithmic changes. One example would be the implementation of ICGS-GMRES to improve ExaWind's solve times.

Other interesting topics that could impact ECP applications, but are currently pursued under SciDAC funding is the development of parallel smoothers that lead to better convergence than Jacobi, such as ILU related methods, as well as the development of multigrid solvers that are more capable to take advantage of the structure of a problem.

4.3.3 The Flexible Computational Science Infrastructure (FleCSI) Project

Overview FleCSI[117] is a compile-time configurable framework designed to support multi-physics application development. As such, FleCSI attempts to provide a very general set of infrastructure design patterns that can be specialized and extended to suit the needs of a broad variety of solver and data requirements. Current support includes multi-dimensional mesh topology, mesh geometry, and mesh adjacency information, n-dimensional hashed-tree data structures, graph partitioning interfaces, and dependency closures, e.g., to identify data dependencies between distributed-memory address spaces.

FleCSI also introduces a functional programming model with control, execution, and data abstractions that are consistent with state-of-the-art task-based runtimes such as Legion[118] and Charm++[119, 120]. The FleCSI abstraction layer provides the developer with insulation from the underlying runtime, while allowing support for multiple runtime systems, including conventional models like asynchronous MPI[121]. The intent is to give developers a concrete set of user-friendly programming tools that can be used now, while allowing flexibility in choosing runtime implementations and optimizations that can be applied to architectures and runtimes that arise in the future.

FleCSI uses static polymorphism, template meta-programming techniques, and other modern C++ features to achieve high runtime performance, customizability, and to enable DSL-like features in our programming model. The FleCSI program structure adopts a three-tiered approach: a low-level core library that is specialized by a mid-level layer to create high-level application interfaces that hide the complexity of the underlying templated classes. This structure facilitates separation of concerns, both between developer roles, and between the structural components that make up a FleCSI-based application.

As an example of how this works in practice, consider the FleCSI mesh topology type:

The low-level mesh interface is parameterized by a policy, which defines various properties such as mesh dimension, and concrete entity classes corresponding to each topological domain and dimension. The mesh policy defines a series of tuples in order to declare its entity types for each topological dimension and domain, and select connectivities between each entity. FleCSI supports a specialized type of localized connectivity called a *binding*, which connects entities from one domain to another domain.

FleCSI separates mesh topology from geometry, and the mesh–from the topology's perspective–is simply a connected graph. Vertex coordinates and other application data are part of the *state model*. Our connectivity computation algorithms are based on DOLFIN[122]. Once vertices and cells have been created, the remainder of the connectivity data is computed automatically by the mesh topology through the following three algorithms: *build*, *transpose*, and *intersect*, e.g., *build* is used to compute edges using cell-to-vertex connectivity and is also responsible for creating entity objects associated with these edges. From a connectivity involving topological dimensions $D_1 \rightarrow D_2$, transpose creates connectivity $D_2 \rightarrow D_1$. Intersect, given $D_1 \rightarrow D'$ and $D' \rightarrow D_2$, computes $D_1 \rightarrow D_2$.

The low-level mesh topology provides a set of iterable objects that a mid-level specialization can make available to an application to allow, at a high-level, iteration through connectivities using an intuitive *ranged-based for* syntax, e.g., forall cells c_i, forall edges e_i of cell c_i. Entities can be stored in sets that also support range-based for iterations and enable set operations such as union, intersection, difference, and provide functional model capabilities with *filter*, *apply*, *map*, *reduce*, etc.

Key Challenges As part of the LANL ATDM effort, FleCSI is one component in a rapidly shifting environment of new software and simulation approaches. This poses several challenges. In particular, the fast evolution of both the runtime backends used by FleCSI, and the applications that are built on top of it make development of FleCSI itself a challenge. Because FleCSI is the fulcrum of a complicated co-design process, it is exposed to instabilities and design challenges from above and below.

FleCSI also faces the challenge of developing a programming model that can span the diverse set of system and node-level architectures for planned and current DOE procurements. Although there is a consistent theme of increased parallelism and scale, the details of individual processor and accelerator architectures present subtly different models of fine-grained parallelism that prove challenging to abstract.

Finally, FleCSI is also seeking to develop fundamental data structures and abstractions that will enable a new and sustainable development process that also satisfies the scope of the LANL ATDM project. This requires careful investigation of requirements and interface design that are essentially collaborative, and reach across institutional and cultural barriers within our organization. These relationships are often challenging to

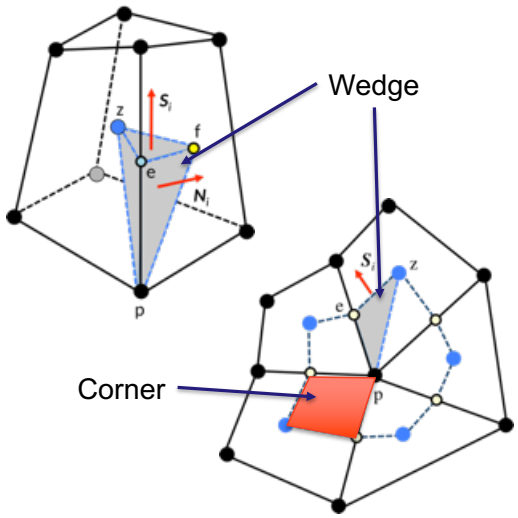


Figure 38: FleCSI unstructured mesh example from the FleCSALE application.

negotiate, and require maturity and a broad knowledge base.

Solution Strategy Our general strategy is one of communication and co-design, whereby, we work in carefully constructed, multi-disciplinary teams to identify gaps in the FleCSI model, design and implement abstractions to fill these gaps, and then verify them in compact applications. To address the challenge of operating in a fluid design environment, we often freeze several components of the application and/or backend runtime to isolate a particular area for refactoring or enhancement. This approach requires a modular design, which is one of the cornerstones of the FleCSI project, and one that has been successfully exercised in several instances.

Recent Progress Recent progress on the FleCSI project has seen the addition of several new storage classes for representing unstructured and sparse data. In particular, we have added *sparse* and *ragged* storage classes. The sparse storage class provides an interface to logically sparse data, e.g., that might arise in the representation of multi-material models or sparse matrices. The ragged storage type is similar, but does not have the notion of columnar indices. Both of these storage types allow dynamic resizing and mutability of the sparsity structure of the represented data. Additionally, we have added *global* and *color* storage classes that allow the representation of simulation state that is either a singleton, or that is on a per-color (the task version of an MPI rank) basis, respectively.

Other recent enhancements include a simplified and improved interoperability interface for managing and interacting with MPI tasks, an improved futures interface, and C++ language extensions for fine-grained data parallel operations.

Next Steps Future work will include the design and implementation of a *set topology* data structure that is suitable for representing several different classes of particles, e.g., *particle-in-cell (PIC)* method, *material-point method (MPM)*, and *Monte Carlo (MC)* methods. We are also working to incorporate changes to the Legion programming model that will provide: a more formal interface for reasoning about and managing graph coloring *dependent partitioning*, and an improved task model that will significantly increase scalability *control replication*. Control replication is an important design improvement to Legion, as it will allow runtime management of dynamically changing data types in a more efficient manner than is currently possible. We will continue to engage in co-design with runtime and application developers to refine and improve the FleCSI model and interface.

4.3.4 LLNL ATDM Math Libraries

Overview The LLNL ATDM Mathematical Libraries project performs work in the MFEM library [123] that 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 [124], 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 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.

Key Challenges 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.

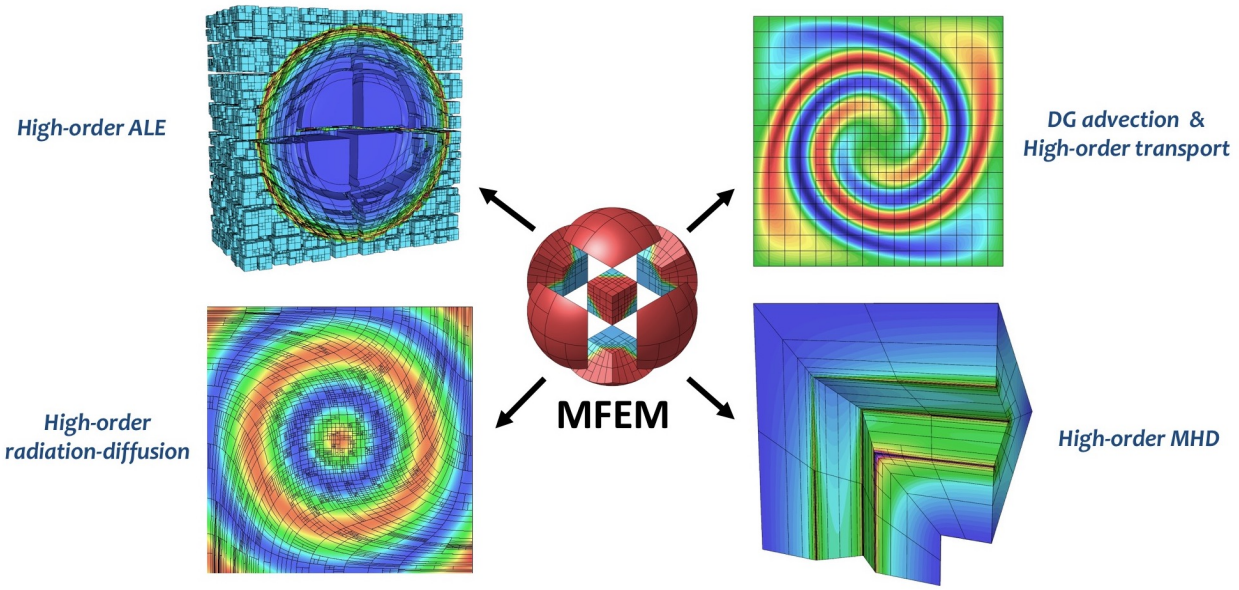


Figure 39: AMR implementation in MFEM allows many applications to benefit from non-conforming adaptivity, without significant changes in their codes.

Scalable algorithms for unstructured adaptive mesh refinement. Adaptive mesh refinement is a common way to increasing application efficiency in problems with localized features. While block-structured AMR has been well-studied, applying AMR in unstructured settings is challenging, especially in terms of derefinement, anisotropic refinement, parallel rebalance and scalability.

GPU porting of finite element codes. 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.

Solution Strategy 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 [125, 126, 127, 128, 129, 130, 131, 132]. 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 39.

As part of the efforts in the ECP co-design Center for Efficient Exascale Discretizations (CEED), the MFEM team is also developing mathematical algorithms and software implementations for finite element methods that exploit increasingq on-node concurrency targeting multiple complex architectures (e.g. GPUs). This work includes the libCEED low-level API library, the Laghos miniapp, and several other efforts available through CEED.

To reach its many customers and partners in NNSA, DOE Office of Science, academia and industry, the MFEM team delivers regular releases on GitHub (e.g. mfem-3.3 and mfem-3.3.2 in 2017) 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.

Recent Progress Selected recent highlights:

- New optimized version of parallel AMR in MFEM, including construction of parallel interpolation, parallel refinement and rebalancing, demonstrated excellent weak and strong scalability during runs on the full Vulcan BG/Q machine at LLNL (400K MPI tasks).
- An interface for the new TMOP mesh optimization capabilities from mfem-3.3.2 was completed in the BLAST code, so that its remesh phase can use the TMOP functionality.
- Completed initial version of multi-group radiation-diffusion in BLAST.
- Participated in the ATDM L2 milestone for MARBL, "Demonstration of Modular Transport Capability in a Multi-Physics Code", which included MFEM support for high-order (HO) to low-order-refined (LOR) field transfer.
- BLAST's closure model was improved by separating the material energies into finite element and point-based parts, avoiding L2 projections of highly oscillatory modes.
- MFEM v3.3.2 was released with many new features including: support for high-order mesh optimization, xSDK support, integration with STRUMPACK, 5 new examples and miniapps, physical-to-reference space mapping, continuous integration testing on Linux, Mac and Windows, and more.
- MFEM v3.3 was released with many new features including: support for PETSc, SUNDIALS, CMake, matrix-free preconditioning, parallel mesh format, 36 new integrators, 16 new examples, and more.
- Moved main MFEM development to GitHub, including many internal branches and pull requests.

Next Steps Our next steps include: prepare and release mfem-3.4; continue to attend and contribute to the weekly MARBL and BLAST meetings; complete an initial draft of MFEM's unified interface extensions to support GPUs and other accelerators; and provide support for the upcoming MARBL L2 milestones.

4.3.5 ATDM SNL Math Libraries

Overview The major project outcome for the SNL ATM Math Libraries project is to provide re-usable and convenient, math-related tools for component-based software engineering of Exascale apps.

This project develops and integrates scalable, modular, and cross-cutting software infrastructure components for ATDM and other future Exascale applications that utilize, where appropriate, ATDM Core CS components: Kokkos performance portability, Sacado/ROL embedded sensitivity analysis and optimization technology, DARMA asynchronous many-tasking, and DataWarehouse. These components include:

1. KokkosKernels (KK): performance-portable sparse/dense linear algebra and graph kernels that utilize the hierarchical memory subsystem expected in future architectures;
2. Scalable Solvers (SS): optimal linear solver algorithms that exploit fine-grain parallelism for vector/SIMT and thread scaling and leverage next-generation execution and communication capabilities;
3. Agile Components (AC): tools for interface abstractions, discretization, time integration, and solution of nonlinear PDEs; and
4. Embedded Analysis (EA): tools for enabling advanced analysis workflows, focusing on embedded sensitivity analysis and optimization with use of derivatives for uncertainty quantification.

This project combines algorithmic R&D with delivery of interoperable software components that are expected to be crucial capabilities that will enable Sandia's ATDM application codes to be performance portable across next-generation computing architectures such as GPUs and Xeon Phis. This work will include integration of these components into the application codes and improving their design and interfaces for mission relevant use cases.

Key Challenges There are several challenges associated with the work this project is conducting. Part of the complexity arises because profiling tools are not yet full mature for advanced architectures and in this context profiling involves the interplay of several factors which require expert judgment to improve performance. Another challenging aspect is working on milestones that span a variety of projects and code bases. There is a strong dependence on the various application code development teams for our own team's success. In addition, we face a constant tension between the need for production ready tools and components in a realm where the state-of-the-art is still evolving.

Solution Strategy To address the challenges above, the SNL ATM Math Libraries project 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.

Recent Progress In this section we provide several high-level snapshots of recent progress in the Math Libraries project

- The EA team has completed the integration of sensitivity analysis and ROL optimization tools with Tempus in support of transient full-space optimization.
- The KK team completed development of a prototype for unified SIMD types as Kokkos SIMD type
- The KK team built batched BLAS kernels based on the Kokkos SIMD type to be run on GPUs. On a P100 GPU, the Kokkoskernels triangular solve and dense matrix-matrix multiply provide up to 2x and 170x speedup, respectively, compared to NVIDIA?s cuBLAS.
- Profiling of the miniEM application in Panzer by the SS team, using a mesh provided by EMPIRE developers, has identified memory growth in problems during setup which has led to a reduced memory footprint.

- Initial results, gathered by the SS team, for performance of Tpetra stack (Belos,MueLu,Ifpack2) have been gathered for the solve phase on GPUs. This resulted in the identification of performance bottlenecks that either are in the process of or have been resolved.
- The AC team completed construction and testing of the operator-split stepper in Tempus. This is required for time integrating the PIC and E-field solves.
- The AC team implemented and verified a variable time-stepping algorithm (Denner, 2014) within Tempus for use with the BDF2 time-integration scheme and other Tempus schemes.

Next Steps The following list details the next steps being taking by each component of the ATDM SNL Math Libraries project:

- The SS team is investigating, in collaboration with the University of Wyoming, the use of Multilevel solvers, which will eventually get integrated into SPARC via MueLu.
- The KK team will continue to work on Integration of Kokkoskernels into IFPACK2 line preconditioners as well as testing on Volta GPUs.
- The AC team will be focusing on the integration of Tempus into EMPIRE-PIC.
- The EA team will extend the work that was done for adjoint sensitivities to transient, full-space optimization.

4.3.6 Enabling Time Integrators for Exascale Through SUNDIALS

Overview This project is enhancing the SUNDIALS library of numerical software packages for integrating differential systems in time using state-of-the-art adaptive time step technologies for use on exascale systems. Through software infrastructure developments, this project is enabling the efficient and robust SUNDIALS time integrator packages to easily interoperate with external linear and nonlinear solver packages developed for exascale computing. In addition, this project is providing a many-vector capability so that SUNDIALS time integrators can more easily operate on data divided over heterogeneous architectures. 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.

Efficient time integrators are essential for ECP because they are at the core of every time-dependent simulation application. However, many applications do not use state-of-the-art methods, and if they do, they often do not yet use them fully on their systems. For example, at the start of the ECP the astrophysics code, Nyx, used an adaptive integration package for solving individual reactions. However, by applying a time integration package to a larger reaction system, the code is able to vectorize more of the calculations and get an accurate solution much faster. By allowing for solvers tuned to exascale systems and vectors that are heterogeneous, SUNDIALS will be more applicable for use in multiphysics systems running on exascale platforms.

Key Challenges Current implementations of efficient time integrators face challenges on many fronts. First, integrators typically have treated the full physical problem with a single step size or have relied on low order operator splitting methods to couple physical processes at different time scales. While research is moving forward within the time integration community on methods for multirate systems, the software infrastructure needs to be in place to accommodate these schemes once they are developed. Second, typical integrators operate on problem data in the form of vectors. These operations suffer from low arithmetic intensity, and their efficiency is often memory bandwidth limited. Lastly, implicit integrators, which are required in many exascale systems, require efficient linear and nonlinear solvers to be highly effective. In addition, by applying integrator-dependent controls on these solvers, their efficiency can be significantly increased. Applying these controls, however, often requires that information about the integrator and its progress be passed to the solver, and software must be designed to effectively pass that information while ensuring adequate encapsulation to provide ease of maintenance and software extension.

Solution Strategy This project includes a number of implementation activities that will prepare the SUNDIALS suite of time integrators for exascale systems. The main activity is a redesign of all linear solver interfaces and encapsulation of the nonlinear solvers within the time integrators in SUNDIALS. The new linear solver interfaces will make it much easier to interface external solver packages while maintaining the efficiency of SUNDIALS integrators. Encapsulating the nonlinear solvers, will reduce redundant code and allow the time integrators to better leverage common code thus lowering the code maintenance burden with SUNDIALS. In addition, the integrators will be able to take advantage of outside nonlinear solvers.

This project is also introducing a set of optional fused vector kernels into SUNDIALS. The goal of these kernels is to execute multiple vector operations at once thereby reducing the number of kernel launches in GPU environments and also reducing the number of communications required for reduction operations. These new kernels will be added to all supplied SUNDIALS vectors and will be invoked through optional interfaces.

Lastly, this project is developing a many-vector capability for SUNDIALS. Due to the tight data encapsulation within SUNDIALS, users are able to supply any vector they would like underneath the integrators. This project will supply the infrastructure needed to make it easy to place a vector of vectors underneath the integrators. This vector of vectors is essential for later implementation of time integrators that will advance various parts of the system with different time step sizes. This many-vector capability will also ease the use of different programming environments as differing vectors can be instantiated on different parts of a hybrid machine.

Recent Progress In September of 2017, SUNDIALS 3.0.0 including new linear solver and matrix interfaces, was released [133]. Figure 40 shows the new organization of SUNDIALS where separate linear solver interfaces are provided for direct and iterative linear solver methods. These interfaces are shared across all SUNDIALS

integrators. Individual integrators have the freedom to supply specific information from the integrator that controls the linear solver. In addition, a single interface to each external solver, such as SuperLU_MT or KLU, is shared across the suite, as opposed to the prior situation where each integrator included its own interface to each solver. In addition, a SUNMATRIX class was developed that can be instantiated with a dense, banded, or sparse matrix. Again, this class is shared by all integrators.

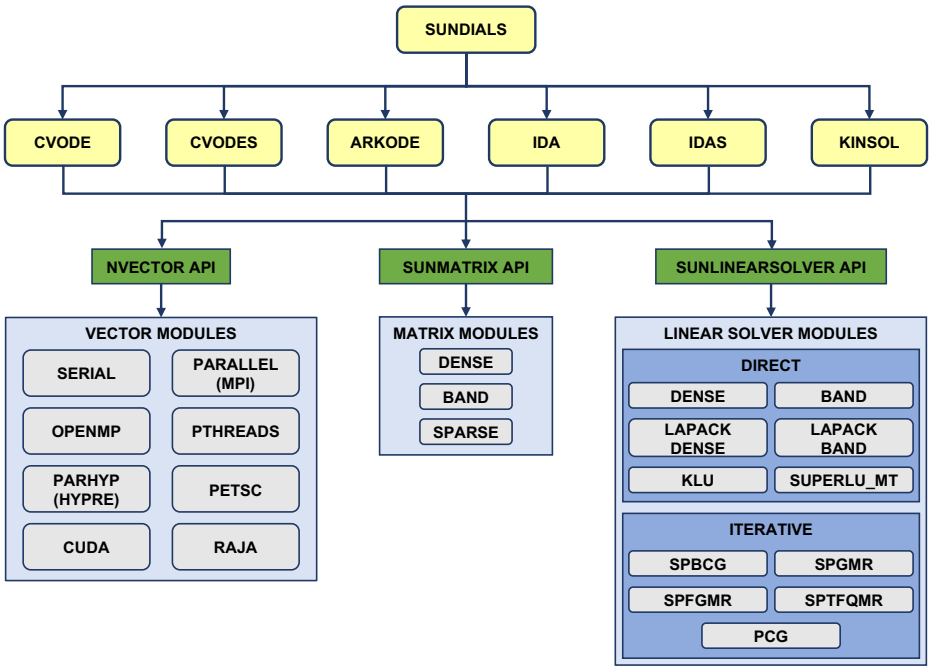


Figure 40: New structure of SUNDIALS showing options for the new SUNLINEARSOLVER and SUNMATRIX classes.

Next Steps During the remainder of FY18, this project team will:

1. Complete a release of SUNDIALS with the new fused vector routines implemented within each supplied vector and optionally used from the integrator packages.
2. Complete a release of SUNDIALS with all nonlinear solvers encapsulated and with implementations of the new nonlinear solver interfaces for both the Newton and fixed point solvers used in the integrator packages.

4.3.7 PETSc-TAO

Overview Algebraic solvers (generally nonlinear solvers that use sparse linear solvers via Newton's method) and ODE/DAE integrators form the core computation of many numerical simulations. No scalable "black box" sparse solvers or integrators work for all applications, nor single implementations that work well for all scales of problem size. Hence, algebraic solver packages provide a wide variety of algorithms and implementations that can be customized for the application and range of problem sizes at hand. PETSc [106, 134] is a widely used software library for the scalable solution of linear, nonlinear, and ODE/DAE systems and computation of adjoints (sometimes called sensitivities) of ODE systems. We focus on three topics: (1) partially matrix-free scalable solvers efficiently use many-core and GPU-based systems; (2) reduced synchronization algorithms that can scale to larger concurrency than solvers with synchronization points; and (3) performance and data structure optimizations for all the core data structures to better utilize many-core and GPU-based systems as well as provide scalability to the Exascale.

The availability of systems with over 100 times the processing power of today's machines compels the utilization of these systems not just for a single "forward solve" simulation (as discussed above) but rather within a tight loop of optimization, sensitivity analysis (SA), and uncertain quantification (UQ). This requires the implementation of a new, scalable library for managing a dynamic hierarchical collection of running scalable simulations, where the simulations directly feed results into the optimization, SA, and UQ solvers. This library, which we call libEnsemble, directs the multiple concurrent "function evaluations" through the tight coupling and feedback described above. This work consist of two parts: (1) the development of libEnsemble and (2) the extension of TAO [135] (our PETSc-based scalable optimization library) with new algorithms and software to utilize libEnsemble.

Key Challenges A key challenge for for scaling the PETSc/TAO numerical libraries to Exascale systems is that traditional "sparse-matrix-based" techniques for linear, nonlinear, and ODE solvers, as well as optimization algorithms, are memory-bandwidth limited. Another difficulty is that any synchronizations required across all compute units–for example, an inner product or a norm–can dramatically affect the scaling of the solvers.

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

Solution Strategy To address the scalability of the numerical libraries, we are developing new solvers and data structures including pipeline Krylov methods that delay the use of the results of inner products and norms, allowing overlapping of the reductions and other computation; partially matrix-free solvers using high-order methods that have high floating-point-to-memory-access ratios and good potential to use many-core and GPU-based systems; and in-node optimizations of sparse matrix-matrix products needed by algebraic multigrid to better utilize many-core systems using a thread neutral "bypass MPI" approach, which implements default interprocessor communication using MPI but bypasses the use of MPI in performance-critical regions for higher performance and thereby maintains MPI portability.

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

Recent Progress In the past year, we have released PETSc 3.9 (available at `http://www.mcs.anl.gov/petsc`) that features pipeline Krylov implementations and improved performance on SIMD architectures. Performance at scale on Theta at Argonne for a 2-D reaction diffusion example on a 16384x16384 mesh using a multigrid preconditioner with 6 levels is shown in Figure 41. The SIMD optimization delivers 2x faster matrix multiply, while not obviously slowing down other kernels. The effect of better vectorization is minimal when available memory bandwidth is low.

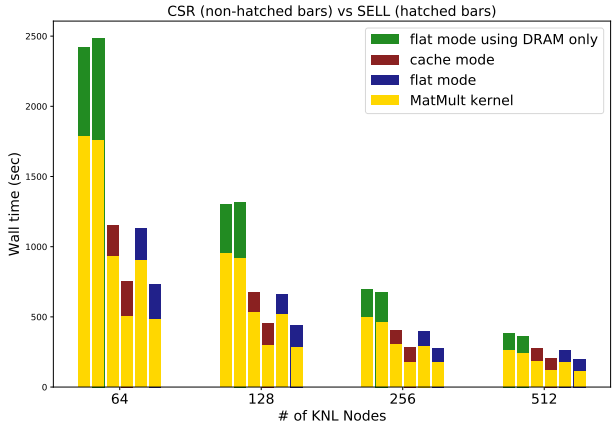


Figure 41: Performance at scale on Theta at Argonne for a 2-D reaction diffusion example on a 16384x16384 mesh using a multigrid preconditioner with 6 levels.

We have also developed the libEnsemble API, implemented it in Python, released version 0.1.0 (available at https://github.com/Libensemble/libensemble), and provided a Spack installation. The code has been testing using the POUNDERS numerical optimization solver.

Next Steps Our next efforts are:

1. **Release libEnsemble with monitoring and pools**: Incorporate improved scheduling and basic online monitoring into libEnsemble with a dynamic pool of simulations. This will be the first supported release.

2. **Release PETSc with enhanced GPU support**: Release of PETSc with enhanced GPU support specifically developed and optimized anticipated architectures.

4.3.8 Factorization Based Sparse Solvers and Preconditioners for Exascale

Overview In this project we will deliver factorization based sparse solvers encompassing the two widely used algorithm variants: supernodal (SuperLU library) and multifrontal (STRUMPACK library). STRUMPACK is further enhanced with scalable preconditioning functionality using hierarchical matrix algebra. Both libraries are purely algebraic, applicable to a large variety of application domains. We will address several challenges that arise in Exascale computing, with the following focus areas: (1) Develop novel approximation algorithms that have lower arithmetic and communication complexity with respect to the input problem size; (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 the higher level algebraic solvers such as hypre, PETSc, Trilinos, and collaborate with ECP application teams for application-specific and hardware-specific tuning of the parameters space to achieve optimal efficiency of our solvers.

Our solver technology is essential for ECP, because many DOE simulation and data analysis codes expected to run on the 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 such as Krylov and multigrid, albeit readily parallelizable on large machines, cannot converge to the solution. The factorization based algorithms being developed in our ECP project 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 iterative methods, or as a coarse grid solver in multigrid methods, just to name a few.

Key Challenges At Exascale we need to address several major challenges: decreasing amount of memory per core, larger impact of communication cost and load imbalance, more heterogeneous memory organization. Our new design of algorithms and codes need to 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 novel factorization methods which require much less memory footprint 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 a communication-avoiding 3D factorization code 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.

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 several areas: (1) develop robust stopping criteria — both absolute and relative — for adaptive (incremental) randomized sampling scheme 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 format, such as HOLDR. The HSS format has asymptotically lower complexity than HOLDR, but has larger prefactor constant. We expect HSS to be more useful for large-scale problems while HOLDR is more useful for mid-range problems; (5) work with the ECP application teams to examine their specific problem characteristics and develop the best clustering/ordering methods to reveal low-rank structures.

Recent Progress We have made significant progress in the following areas:

1. We developed and evaluated a fully algebraic sparse preconditioner in STRUMPACK. On top of the baseline multifrontal direct solver, we use low-rank compression in dense frontal matrices to obtain approximate factorization. We showed that our MF+HSS preconditioner is more robust for numerically hard problems than many alternatives. Our code strong scales to over 6000 cores [136] (Fig. 42).

2. We developed several strategies to enhance scalability of triangular solve in SuperLU_DIST. One is an asynchronous tree-based broadcast/reduction scheme which reduces latency and improves communication load balance. Another is efficient threading implementation and BLAS operations. The new code is 4.4x and 6.1x faster on 4096 cores with one and 50 right-hand sides, respectively [137] (Fig. 43).

3. We developed a new communication-avoiding 3D sparse LU factorization (IPDPS) algorithm that has provably asymptotic lower communication complexity in both latency and volume. The prototype implementation in SuperLU_DIST achieves up to 27x improvement over the baseline 2D algorithm when run on 24,000 cores of Edison at NERSC [138].

4. In collaboration with ExaGraph ECP project, we evaluated the performance of a parallel pre-ordering algorithm called Approximate-Weight Perfect Matching (AWPM) for pivot selection in SuperLU_DIST. For most practical problems (e.g.,DOE apps, and SuiteSparse) the weights of the perfect matchings generated by AWPM often within 99% of the optimum. The MPI+OpenMP implementation on Cori at NERSC scales up to 256 nodes —- 2500x faster than serial MC64, and up to 114x speedup on 256 nodes (17,408 cores) Cori-KNL [139]. The interface to AWPM are already implemented in both STRUMPACK and SuperLU and are released.

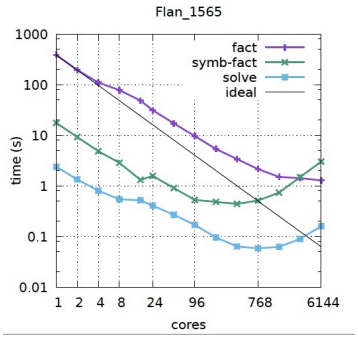


Figure 42: STRUMPACK scaling of three phases

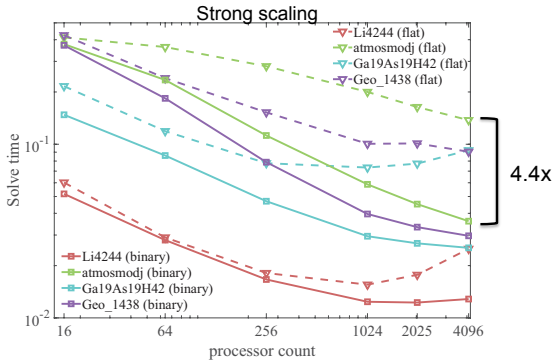


Figure 43: SuperLU triangular solve scaling with 4 matrices.

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

- For STRUMPACK, we will improve the performance of the HSS solve routine, add OpenMP and reduce communication. We will implement the HOLDR low-rank format.
- For both STRUMPACK and SuperLU, we will build detailed performance models and performance specific code optimizations for the ECP applications that use our solvers.

4.3.9 ForTrilinos

Overview The Exascale Computing Project (ECP) requires the successful transformation and porting of many Fortran application codes in preparation for ECP platforms. A significant number of these codes rely upon the scalable solution of linear and nonlinear equations. The Trilinos Project contains a large and growing collection of solver capabilities that can utilize next-generation platforms, in particular scalable multicore, manycore, accelerator and heterogeneous systems. Trilinos is primarily written in C++, including its user interfaces. While C++ is advantageous for gaining access to the latest programming environments, it limits Trilinos usage via Fortran. Several ad hoc translation interfaces exist to enable Fortran usage of Trilinos, but none of these interfaces is general-purpose or written for reusable and sustainable external use.

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. The project uses an interface generator, SWIG, which the project is extending, to create the object-oriented Fotran wrapper code that users can access directly. This language translation will occur in both directions; ForTrilinos will provide an inversion of control functionality that enables custom extensions of the Trilinos solvers that are Fortran-based. Once the ForTrilinos project is complete, a functional, extensible suite of capabilities to access Trilinos on next-generation computing systems will be provided. Several examples of this technology will be demonstrated within ECP codes with the aim of meeting their simulation goals and illustrate the technology to other Fortran-based ECP codes. As a second benefit to ECP from this effort, the Fortrilinos approach of using SWIG to generate interfaces could be extended to other C/C++ based tools and software within the ECP stable.

Key Challenges 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 The ForTrilinos project has two primary thrusts:

1. **SWIG development to support Fortran wrapping:** The new Fortran module for SWIG allows for automatic generation of interfaces.
2. **Incremental wrapping of selected Trilinos functionality:** ForTrilinos provides robust interfaces to selected Trilinos capabilities, including distributed data objects, linear and nonlinear solvers.

ForTrilinos started with developing a Fortran module for the Simplified Wrapper Interface Generator (SWIG) utility [140]. SWIG is used to parse C++ code and generate wrapper code, and was already used for this purpose for several dozen target languages, most notably Python. The project took the approach of adding a Fortran 2003 wrapper generator for SWIG in order to fulfill many of the critical feature requirements for ForTrilinos.

The developed SWIG/Fortran functionality allowed to proceed with automatic generation of Fortran interfaces to selected Trilinos libraries. The work is conducted in phases, with each phase increasing the number of wrapped of Trilinos packages.

The first phase, completed in the first year, developed interfaces for a critical set of features required for solving linear and eigen-problems. The second phase addresses nonlinear solver, including developing automatic approach for inversion-of-control capability. Finally, in the third phase, the challenge of interoperable heterogeneous data passing between Fortran and C++, including GPU data, is addressed, including performance testing.

Recent Progress Figure 44 illustrates a new Inversion-of-Control (IoC) implementation in ForTrilinos. The new approach allows Fortran users to define an operator by using a derived type on the Fortran side, and use Trilinos algorithms, e.g. Krylov solvers, to solve the linear system with that operator.

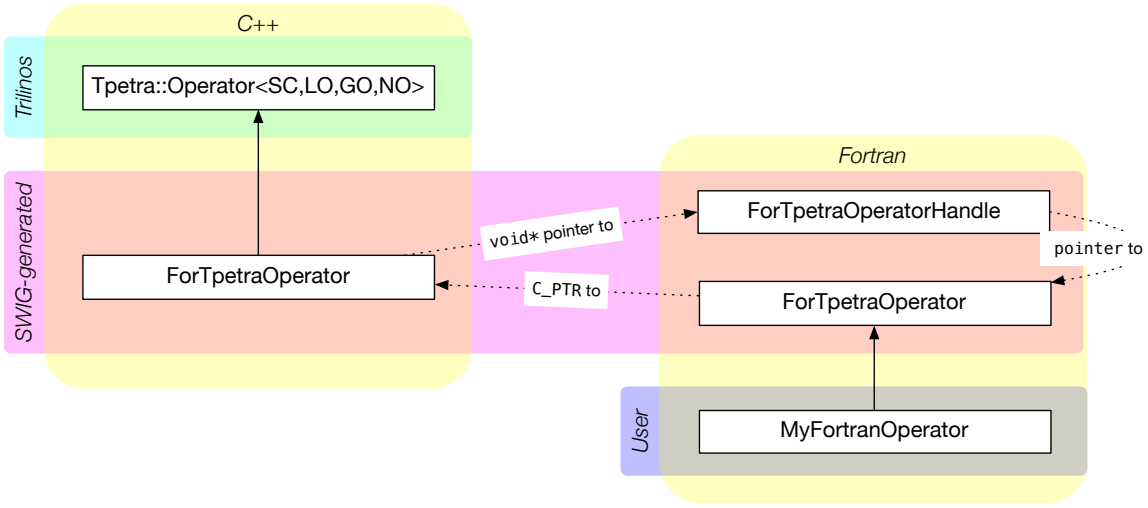


Figure 44: The proposed Inversion-of-Control approach allowing Fortran applications to define operators on Fortran side while still using ForTrilinos types.

This approach allows the user callback functions to interact with native Fortran types and ForTrilinos class wrapper types. In the same vein, users would not have to manually pass `type(C_PTR)` instances into and out of the callback function, as the C++ Fortran conversions can be tedious and error-prone, which indeed is the motivation for using SWIG to generate ForTrilinos. Another important feature is that it allows the application code to extend Trilinos without having to generate any new interface code, either by hand or using SWIG. In other words, the Fortran end user should not have to know C++ or SWIG.

Next Steps Our next efforts include:

1. **Merge Fortran module to upstream SWIG:** The work is in progress to push the developed SWIG/Fortran functionality to upstream SWIG project. This would allow projects both inside and outside ECP ecosystem to start using this functionality independently of ForTrilinos to provide Fortran interfaces to their own libraries. We anticipate multiple projects taking advantage of this capability.

2. **Provide wrappers for more libraries:** ForTrilinos will continue efforts to increase the number of wrapped Trilinos libraries. The next phase of the project will include libraries corresponding to nonlinear solvers, such as NOX. We anticipate that once that functionality is available, many applications in ECP, including E3SM-MMF, will be able to start using ForTrilinos in the production.

3. **Provide interfaces for heterogeneous platforms:** ForTrilinos will develop support for heterogeneous memory through providing access to Kokkos-based interfaces in Trilinos. This will allow full exposure to Trilinos capabilities targeting Exascale machines.

4.3.10 SLATE

Overview The objective of the Software for Linear Algebra Targeting Exascale (SLATE) project is to provide fundamental dense linear algebra capabilities to the US Department of Energy (DOE) and to the high-performance computing (HPC) community at large. To this end, SLATE will provide basic dense matrix operations (e.g., matrix multiplication, rank-k update, triangular solve), 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. However, after two decades of operation, ScaLAPACK is past the end of its lifecycle and overdue for a replacement, as it can hardly be retrofitted to support hardware accelerators, 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, many-node HPC machines with large numbers of cores and multiple hardware accelerators per node. For typical dense linear algebra workloads, this means getting close to 90% of the theoretical peak performance and scaling to the full size of the machine (i.e., thousands to tens of thousands of nodes). This is to be accomplished in a portable manner by relying on standards like MPI and OpenMP.

SLATE functionalities will first be delivered to the ECP applications that most urgently require SLATE capabilities (e.g., EXAALT, NWChem, QMPACK, GAMESS, and CANDLE) and to other software libraries that rely on underlying dense linear algebra services (e.g., FBSS). SLATE will also fill the void left by ScaLAPACK's inability to utilize hardware accelerators, and it will ease the difficulties associated with ScaLAPACK's legacy matrix layout and Fortran API. Figure 45 shows SLATE within the ECP software stack.

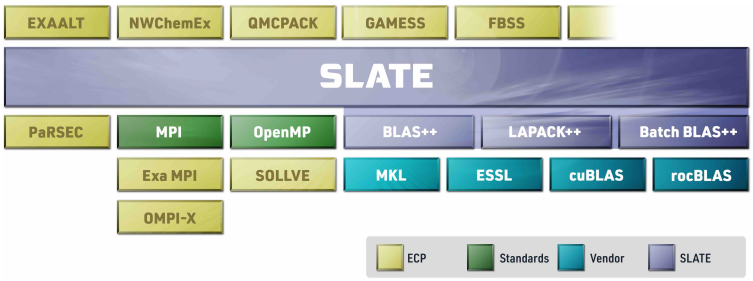


Figure 45: SLATE in the ECP software stack.

Key Challenges

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

2. **Aiming at a hard hardware target:** SLATE's acceleration capabilities are being developed in an unforgiving hardware environment, where the computing power of the GPUs exceeds the computing power of the CPUs, as well as the communication capabilities of the network, by orders of magnitude.

3. **Using cutting-edge software technologies:** Finally, SLATE is being designed using cutting-edge software technologies, including modern features of the C++ language, as well as fairly recent extensions of the OpenMP standard and the MPI standard.

Solution Strategy

1. **Deliberate design phase:** The need for building SLATE from scratch was a primary focus of SLATE's architecture and design in the initial project milestones. First, we conducted a careful analysis of all the relevant technologies—existing and emerging [141]. Then we designed the initial architecture [142].

Going forward, the development process is based on alternating feature development with refactoring and redesign of the underlying infrastructure.

2. **Accelerator-centric focus:** Hardware accelerators (e.g., graphic processing units [GPUs]), are treated as first-class citizens in SLATE, and accelerated performance is the primary focus of the performance engineering efforts. Device performance relies on highly optimized routines in vendor libraries, mostly the batch matrix multiply routine (i.e., batch gemm). Care is also taken to hide accelerator communication in addition to hiding message passing communication. To address the network deficiencies in the long term, the team is investigating cutting-edge algorithmic solutions like communication-avoiding algorithms [143] and randomization algorithms [144].

3. **Community engagement:** The SLATE team maintains regular contact with the OpenMP community, the ECP SOLLVE project, and the broader OpenMP community (joining the OpenMP ARB). The team also engages vendors and has contacts at Intel, NVIDIA, and AMD. The SLATE team is co-located with the ECP OMPI-X project and has a direct line of communication with the MPI community.

Recent Progress The most recent development in SLATE is the implementation of all level-3 PBLAS routines, covering all four precisions, single real (S), single complex (C), double real (D), double complex (Z), and all combinations of input parameters, `side`, `uplo`, and `trans`. SLATE's accelerated runs deliver up to a 20× performance improvement over ScaLAPACK multi-core runs on the SummitDev machine at the Oak Ridge Leadership Computing Facility [1] (Figure 46). A more exhaustive performance analysis is available in "SLATE Working Note 5: Parallel BLAS Performance Report," [145].

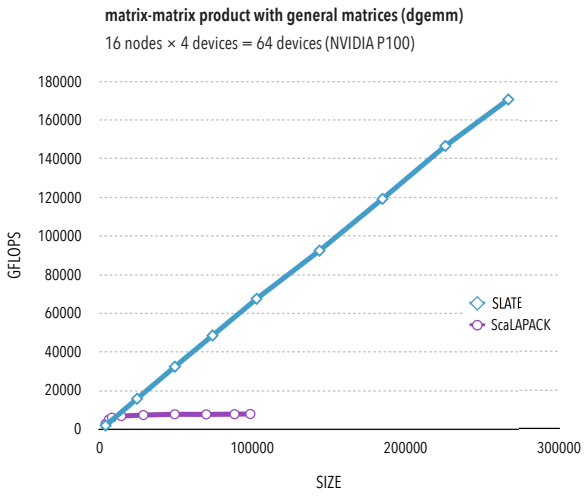


Figure 46: Accelerated performance using SLATE compared to multi-core performance using ScaLAPACK.

Next Steps

1. **Matrix norms:** We plan to implement matrix norms by the end of June 2018. This includes routines for computing the one norm, max norm, infinity norm, and a specialized routine for computing a column-wise norm, which is required for mixed-precision linear solvers based on iterative-refinement.

2. **Linear solvers:** We plan to implement linear solvers by the end of September 2018. This includes an LL^T factorization, an LU factorization, an LDL^T factorization, and the corresponding forward/backward substitution routines.

[1] https://www.olcf.ornl.gov/kb_articles/summitdev-quickstart/

3. **Least squares solvers:** We will implement least squares solvers by the end of December 2018, including the QR and LQ factorizations and the routines for the generation and application of the Q matrices.

All four standard ScaLAPACK precisions will be covered, and all routines will support hardware acceleration.

4.3.11 PEEKS

Overview The PEEKS project 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 are turned into production-quality and integrated into the ECP software ecosystem as part of the Trilinos and MAGMA-sparse software stack. As leading developers of these two software packages (Trilinos and MAGMA-sparse), the members of the PEEKS project have a strong record of delivering software packages that are used on future leadership-class supercomputers. In fact, ECP application projects submitted to the DOE explicitly mention MAGMA and Trilinos as part of the software ecosystem to utilize Exascale computers.

In order to make the new algorithms easily accessible to ECP applications, we design a generic software interface for the parallel preconditioning techniques in MAGMA-sparse and the communication-avoiding Krylov solvers. Featuring these interfaces, we integrate the production-ready solvers and preconditioning techniques into the ECP software stack via the xSDK4ECP software effort. This enables the ECP applications to adopt the new technology without fundamental changes in their application design.

A particular focus of the PEEKS project is on software portability, and we aim to deliver high performance across the list of ECP-supported Exascale architectures. This effort is realized using sophisticated software development and testing practices, ensuring a robust production-quality software that can be relied upon by ECP application projects.

Key Challenges Running iterative solvers on the next-generation HPC systems results in two 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.

Both challenges require the redesign of existing iterative solvers with respect to higher parallelism within all building blocks, and a reduced number of communication and synchronization points. In the last few decades, numerous efforts have investigated the potential of communication-avoiding (CA) and pipelined Krylov solvers [146, 147]; however, the implementations usually remained in prototype status and rarely made it into production code. Similarly, significant effort was put into developing new preconditioning techniques that allow for the efficient parallelization of the preconditioner setup and the preconditioner application [148, 149, 150]. Again, most implementations were experimental and rarely adopted by application code.

Turning the experimental code into production quality alone is rarely sufficient to make applications adopt the new algorithm technology, as application projects usually refrain from refactoring their code to integrate new solvers or preconditioners. This motivates a second step that is of comparable importance: the design of generic interfaces that allow for easy exchange of solver and preconditioner infrastructure without modifying the application code.

An important aspect for complex application codes to possess is the quick and comprehensive identification of performance bottlenecks. Against the background of integrating novel solver and preconditioning techniques, this requires monitoring algorithm-specific counters that provide insight about the appropriateness of certain parameter choices. Manual code instrumentation is cumbersome and practically prevents the parameter tuning in complex application codes. This motivates exposure of "software-defined events" (SDE) to external profiling software that is available on the target hardware.

Solution Strategy The primary thrusts of the PEEKS project are:

1. **Generic numerical software interface design:** To propel the painless integration of new solver and preconditioner technology, we design a generic interface that adheres to Better Scientific Software (BSSw) design principles [151]. The purpose of a generic interface is not restricted to the ECP effort; it is anticipated to be adopted by other software efforts in the scientific computing community.

2. **Pipelined and CA Krylov methods:** We realize pipelined and communication-avoiding Krylov methods in production-quality code. We provide interfaces and support their integration into existing application codes.

3. **Parallel preconditioning techniques:** We implement the parallel generation and application of popular preconditioning techniques in production-ready code. The preconditioners are capable of leveraging the parallelism available on a single node (including many-core accelerators) and allow for easy integration into application codes.

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

Recent Progress

1. We populated a white paper on a generic software interface in the scientific community and collected feedback including design modifications. The white paper is accessible at the PEEKS project webpage [152].

2. We recently completed the production code implementation of the parallel generation of preconditioners based on incomplete factorization. In an initial assessment of the hardware anticipated to become the backbone of the Summit and Sierra pre-Exascale platforms, we identified for benchmark test problems significant advantages over vendor-tuned functionality (see Figure 47).

3. Recently, we also realized pipeline Krylov solvers in the Trilinos production code ecosystem and analyzed the performance for ECP application use cases. A performance assessment report is available on the PEEKS webpage [152].

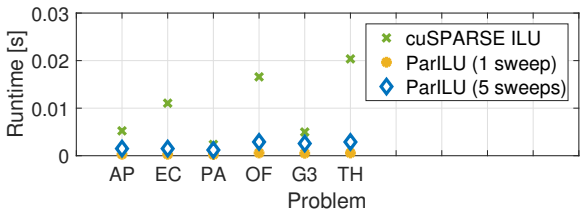


Figure 47: Runtime of the ILU preconditioner generation on an NVIDIA V100 GPU using the novel ParILU algorithm realized in the PEEKS project and NVIDIA's cuBLAS routine, respectively.

Next Steps Our next efforts are:

1. **Integrate novel algorithm functionality into ECP application projects:** We are in contact with domain scientists that are willing to adopt the developed pipelined Krylov solvers and parallel precondition techniques to serve as production code assessment.

2. **Integrate the software-defined events:** Together with the Exa-PAPI team, we already explored the potential of software-defined events (SDE) for numerical LA libraries. We next integrate these into production code and assess the usability for ECP applications.

3. **Parallel preconditioner application:** With the parallel preconditioner generation being resolved, we now focus on efficient preconditioner application using the parallelism available within a single node. This step is strongly tied to the SDE effort, as we expect significant benefits for application-tuned parameter choices in particular.

4.3.12 ALExa

Overview The ALExa project (*Accelerated Libraries for Exascale*) focuses on preparing the DTK and Tasmanian libraries for exascale platforms and integrating these libraries into ECP applications. These libraries deliver capabilities identified as needs of ECP applications: (1) the ability to transfer computed solutions between grids with differing layouts on parallel accelerated architectures, enabling simulation projects such as ExaAM and ExaSMR to seamlessly combine results from different computational grids to perform their required simulations (DTK); and (2) the ability to construct surrogate models with low memory footprint, low cost and optimal computational throughput, to enable the use of methods for optimization and uncertainty quantification for large scale engineering problems, as well efficient multi-physics simulations in projects such as ExaStar (Tasmanian).

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.

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-free interpolation capabililities:* multivariate data interpolation between point clouds; compactly supported radial basis functions; nearest-neighbor and moving least square implementations; common applications include conjugate heat transfer, fluid structure interaction, and mesh deformation. *Performace portable search capabilities:* threaded and GPU implementations of spatial tree construction; threaded and GPU implementations of various spatial tree queries; MPI front-end for coordinating distributed spatial searches between sets of geometric objects with different decompositions; communication plan generation based on spatial search results. *URL:* https://github.com/ORNL-CEES/DataTransferKit

Tasmanian (Toolkit for Adaptive Stochastic Modeling and Non-Intrusive Approximation) *Purpose:* Constructs surrogate models for high dimensional problems to help solve large scale engineering and multiphysics problems *Significance:* Applications may require reduced representations of higher dimensional data, for example, to optimize storage of high dimensional tabular data or optimally perform UQ calculations. In addition, Tasmanian offers capabilities in inverse problems, e.g., calibration and inference. *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

Key Challenges

DTK: The general case of transferring data between grids of unrelated applications requires an all-to-all communication. This is increasingly challenging as communication to computation ratios are decreasing on successive HPC systems. Furthermore, the gridcell search procedures to locate interpolation points require tree search methods difficult to optimize on modern accelerated architectures due to vector lane or thread divergence. Also, maintaining high accuracy for the transfer requires careful attention to the mathematical properties of the interpolation methods and is highly application-specific. Unlike other, "black box" libraries, grid transfer methods may require close collaboration with application partners to understand customer requirements (accuracy and conservation law requirements, distribution of grids, location and layout of grid data in memory hierarchy, etc.)

Tasmanian: Sparse grid methods require dense linear algebra operations with matrix sizes that may be large. These operations must be well-optimized for the target architectures, including accelerated nodes. Uncertainty quantification methods can require many runs of an application as input to forming a surrogate model, and the runs may have very different runtimes, creating a scheduling problem that must be solved in order to maximize performance.

Solution Strategy

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: To support performance portability across platforms for dense linear algebra operations, a collaboration with the SLATE ECP project is underway to take advantage of the new library software being developed. A DAG scheduler will be implemented under the project to enable optimal execution of multiple application instances for UQ problems.

Recent Progress

DTK: Ongoing work with a growing list of ECP application partners is continuing. Extensive work has been done to define the Fortran, C and C++ DTK interfaces to support partner projects. A set of benchmark problems has been developed to guide DTK method and code development. Recent progress has been made to optimize the search methods for modern-generation GPUs.

	build	speedup	knn search	speedup	radius search	speedup
boost rtree	.267s	(1.26)	3.05s	(2.67)	.148s	(3.03)
nanoflann kdtree	.376s	(0.90)	4.46s	(1.82)	.168s	(2.67)
DTK (serial)	.337s	(1.00)	8.13s	(1.00)	.448s	(1.00)
DTK (omp 2 threads)	.197s	(1.71)	4.14s	(1.96)	.264s	(1.70)
DTK (omp 4 threads)	.115s	(2.93)	2.31s	(3.52)	.164s	(2.73)
DTK (omp 8 threads)	.088s	(3.83)	1.31s	(6.21)	.097s	(4.62)
DTK (omp 16 threads)	.055s	(6.13)	0.80s	(10.2)	.072s	(6.22)
DTK (CUDA Summitdev)	.045s	(7.49)	4.51s	(1.80)	.014s	(32.0)

Figure 48: DTK tree search, OpenMP and Summitdev GPU speedups

Tasmanian: The infrastructure of Tasmanian has been upgraded to support the broader ECP focus of the work. GPU acceleration of sparse grid surrogates has been implemented. Tasmanian recently enabled the ExaStar project to reduce the size of a large-memory table of neutrino opacities by 100X while still preserving accuracy.

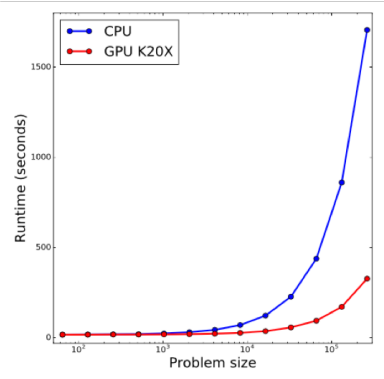


Figure 49: Tasmanian speedup on Titan node using cuBLAS

Next Steps

DTK: A major focus for the near future is to optimize performance for the targeted architectures using the developed benchmark cases. This will be followed by work on a showcase problem to demonstrate performance.

Tasmanian: Work will continue to integrate the SLATE dense linear algebra capabilities into Tasmanian to enable high performance and performance portability. Asynchronous sampling using DAG methods will be implemented for the sparse grid methods.

4.4 DATA & VISUALIZATION

This section present projects in Data & Visualization.

4.4.1 Data & Visualization Software Development Kits

Overview The Data & Visualization SDK effort is focused on identifying meaningful aggregations of products in this technical area. SDK efforts are in the early stages of planning and execution. Most of the work on SDKs has been driven from the SW Ecosystem & Delivery technical area. A description of the SDK effort can be found in Section 4.5.1.

4.4.2 2.3.4.02 LANL ATDM Data and Visualization

Overview The LANL ATDM Data and Visualization project is focused on developing scalable systems software for the generation, analysis, and management of data produced by ECP applications. This project is essential for ECP because existing systems software is inadequate with respect to deploying advanced data collection and analysis capabilities into HPC data centers. Existing software cannot leverage the enormous performance and data capacity provided within Exascale data centers such that scientists can effectively generate insight using the data generated by extreme scale simulations.

Scientific simulations running on Exascale platforms will continue to have access to enormous solid-state storage tiers that provide opportunities for rapid data acquisition. Similarly, massive campaign storage systems built from affordable media offer the opportunity for maintaining large data sets over longer time periods to support longer duration simulations and accompanying analysis. Finally, advanced monitoring frameworks built using time-series databases and analysis storage systems are deploying within HPC data centers. Each of these systems requires significant systems software development and integration efforts to create new opportunities for data management within scientific applications, analysis codes, and HPC facilities. Upon completion of our project, domain scientists will leverage these new capabilities to improve the time to insight for scientists using extreme scale scientific simulations.

Key Challenges Re-architecting storage systems to impact both applications and facilities is challenging both in its breadth and depth. Fundamentally, our approach is focused on unlocking the value that currently exists within scientific data, but has traditionally been too time consuming to extract. As part of these efforts we've identified the need for software and facility infrastructure that supports finding data within the diverse set of storage resources

Similarly, our efforts to better provide in-application support for modern analysis techniques require careful attention to performance, memory use, and data reduction without loss of insight. Scaling challenges.

Solution Strategy The LANL ATDM Data and Visualization ECP project is focused on delivering new systems software capabilities for creating, analyzing, and managing data for Exascale scientific applications and Exascale data centers. We have identified 4 distinct areas that are in need of specific improvements.

MarFS is the only campaign storage system within the DOE complex and is also the only HPC storage system built using scale-out principles with affordable SMR hard drives. The LANL monitoring stack is leveraging available open source software to build dashboards for monitoring both the data center and scientific application performance. Finally, the application level software technologies, Cinema and HXHIM, are 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

Recent Progress The MarFS file system is currently deployed as the campaign storage tier with over 60PB of capacity currently under management in our secure computing environment. Recent progress includes the development of a new highly resilient backend based on nested parity. We have also extended our top-level erasure approach to use RDMA operations for more efficient coding and data movement.

The LANL monitoring stack has been successfully deployed into multiple computing enclaves within LANL's HPC facility. We have procured additional hardware and are currently deploying the monitoring infrastructure and dashboards into LANL's secure computing environment. Our approach currently supports both administrators, analysts, and code teams with multiple dashboards for each role. We continue to refine our security approach to ensure that new monitoring monitoring dashboards comply with the requirements of LANL's CCB, the voting body for ensuring that all LANL deployments are secured appropriately. The use of our monitoring system by application teams (which monitor information that includes classified data) has required a highly granular approach to storage and access roles.

Recent progress on HXHIM, a key-value store for HPC platforms, includes the integration of a new transport layer based on Margo and Mercury (projects under development by the ECP data libs project). The fundamental architecture of HXHIM now leverage new support for using a high-performance RPC package

(Mercury) layered beneath a C++ wrapper for Margo (called Thallium). HXHIM provides bulk (multi-key) primitives that enable efficient use of HPC interconnects and typical scientific storage workloads.

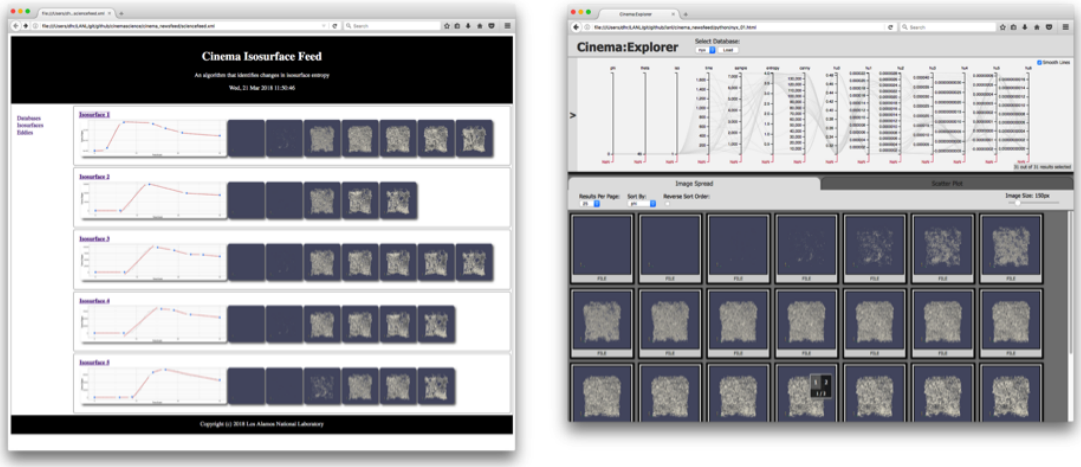


Figure 50: Screen capture of both the Cinema:Newsfeed viewer (left and Cinema:Explorer viewer, showing different views of data from the workflow described above. On the left, the Cinema:Newsfeed viewer shows a graph resulting from the change detection algorithm, and snapshots from the timesteps at the inflection points for the change detection algorithm. This viewer shows a compact 'newsfeed' view of the end-to-end analysis. Clicking on the images in this viewer takes the scientist to a more detailed view of the overall set of features captured during the simulation. Because these viewers are implemented in a browser, it is easy to link different viewers together for new tasks and workflows.

Recent progress on the Cinema project has focused on development of capability and workflows for change detection in-situ data analysis artifacts. These promote new types of analysis of data from ECP simulations, including automated and assisted analysis. We are creating capabilities that allow scientists more options in analyzing and exploring the results of large simulations by provide a workflow that 1) detects features in-situ, 2) captures data artifacts from those features in Cinema databases, 3) promotes post-hoc analysis of the data for things such as change detection, and 4) provides data viewers that allow interactive, structured exploration of the resulting artifacts. In particular, in our most recent milestone, we ran a Nyx simulation and captured in-situ features (isosurfaces), saved a Cinema database to disk, analyzed that database to determine inflection points in the complexity of those isosurfaces, and presented the results in a newsfeed viewer linked to other views of the data. This overall workflow provides a flexible method of applying new algorithms to the analysis and visualization of extreme scale data.

Next Steps We are continuing to add features and work on improving code quality for each of our projects. This includes improving the performance, scalability, and maintainability of each of the associated subprojects. Our MarFS project is focused on improving read/write performance via increased protocol efficiencies. Our monitoring work continues to explore the most efficient indexing for supporting popular queries without dramatically increasing storage requirements. HXHIM is in the process of adding layers for managing memory and network resources efficiently. Cinema is identifying new application workflows that can be reasonably made efficient and new analysis methods to apply efficiently for cinema users.

4.4.3 LLNL ATDM Data & Viz Projects: Workflow

Overview The ATDM Workflow project at LLNL has a long term vision to enable reproducible and pedigreed computational science and introduce advanced analytics and machine learning capabilities to our user community. We are building an ecosystem of re-usable components for user workflows that will enhance the end-to-end productivity of ASC HPC assets and enable users to introduce modern data analytics technologies to their workflows. The Workflow project is focused on three areas prioritized by the LLNL user community: problem setup, simulation management, and data management and analytics. The project focuses on an ecosystem of tools and libraries because there is no single overarching workflow system that can meet the needs of users in the wide variety of domains at LLNL. The tools produced by this project are focused on reducing extrinsic cognitive load, that is, to reduce the bookkeeping and repetitive one-off scripting that is required with current file-oriented workflows. The project seeks to leverage modern data analytics capabilities, by providing tools that users can leverage to inject simulation information into these systems.

Key Challenges The three biggest challenges for implementing improved problem setup, simulation management, and data management and analytics for users are: user community heterogeneity and complexity, security requirements, and user adoption. First, the large number of user domains and the number of largely manually driven workflows in use precluded the installation of a single encompassing workflow management and product lifecycle management capability. Second, the security posture required to install tools at NNSA labs makes it difficult to stand up federated infrastructures and servers. Finally, adoption of tools into existing workflows requires that the tools provide a clear value proposition to the user community, and any tool that requires large changes in existing practice faces an increased chance of being rejected by users. This is particularly true for systems that seek to capture full provenance and pedigree data: they often require centralized servers and that all workflow activities take place in a particular tool or environment.

Solution Strategy We established three sub-projects to focus on aspects of the three areas prioritized by users:

1. **C2C:** the Contours to Codes project focused on building tools for interchanging and streamlining the process of going from documents describing experiments to the corresponding simulation configurations. C2C has adopted a format for 2D geometry defined by Los Alamos National Laboratory, and has built Python and C++ parsers for this toolset with features needed for LLNL applications.

2. **Siboka:** an ecosystem of scientific workflow components to facilitate workflow and data management, increase pedigree/reproducibility, and enable the use of modern analytics technologies. **Siboka** includes a set of Python packages and command line tools for capturing non-bulk data and meta-data from simulation runs, importing data to SQL and Cassandra (a column store), allow querying of data in those backends, and exporting results for use in downstream tools. The core tool being developed is called **Sina**, which enables these operations. In addition, a specification for a JSON schema and toolset called **Mnoda** is being developed to generalize an application specific tool created by the project to gather and query non-bulk data and meta-data from simulation run directories.

3. **Syflux (formerly VV4ALE3D):** a prototype web portal for setting up, running, and managing simulations and their results. The original prototype, VV4ALE3D, is integrated with the Lorenz web infrastructure deployed in Livermore Computing. The VV4ALE3D application is still deployed and hosted on the Livermore Computing web infrastructure. Syflux represents an updated version based on the Django web infrastructure and hosted on project-managed web servers outside the data center authentication realm. The team seeks to develop into a problem setup data portal as well as a way to deploy workflow management tools.

These projects represent composable tools targeting distinct aspect of user pain points. C2C enables interchange of problem setup information across applications, and a format in which to store and curate the results of the time consuming process of setting up new simulations. Siboka presents the users with distinct and focused Python packages that they can include in existing scripts, so that they can incrementally adopt the technologies they see as relevant to their work. Syflux and associated exploration of Jupyter,

represents customizable ways to provide access via web-browser to simulation data, problem setup and related databases, and documents. In this way, the team mitigates risk by providing the user community with a set of components that encompasses best practices, common operations, and next-generation capabilities, allowing the user community to leverage them to build custom workflows where needed while reducing duplicated work.

Recent Progress **C2C** recently released an initial C++ parser to LLNL application codes, a key deliverable to enable adoption of a common 2D geometry format across our user-base. The team has also defined an assembly file specification to allow manipulation and composing of lower level geometric information, and implemented a set of python tools for this. These tools are now being tested by a few initial users and they have demonstrated that the tools can be used for code inter-comparison.

The Siboka team has recently created an importer for *Cassandra*, a scalable, open-source column store used extensively by companies like Netflix and Walmart. The team demonstrated the ability to query and export data from SQL using the **Sina** package, and plans to extend that functionality to the Cassandra back-end. The team is now exploring how *JupyterHub*, *Jupyter Notebooks*, and the newly released (in beta) *JupyterLab* enhance sharing and reproducibility while incorporating these modern database capabilities. The first version of the **Mnoda** schema has been defined, and the Cassandra importer leverages that tool to map simulation meta-data and non-bulk scalars and file paths to the Cassandra and SQL back-ends.

The Syflux effort has stood up a Django-based web portal, and has enabled users to upload and curate data and media files and implemented a basic permission model for them. The ability to cross-link resources (input files, output files, and documents/reports) was also added to the prototype.

Next Steps

1. **C2C:** the highest priority for the team is to integrate the C++ library into the first application code and continue to enhance assembly files.
2. **Siboka:** Release an initial Mnoda schema, and create a specification for a C linkable API for writing data for applications that do not want to write the JSON output directly. Enhance the support for Cassandra and release an updated version of **Sina** to our users, as well a set of JupyterHub examples.
3. **Syflux**: prototype an integration of multiple databases, such as the Granta database into the portal.

4.4.4 SNL ATDM Data and Visualization: IOSS and FAODEL

Overview The SNL ATDM Data and Visualization project is developing data management software to improve how applications store and exchange large datasets efficiently on Exascale platforms. The data portion of this project is composed of two related efforts: (1) production work focused on improving Sandia's IOSS library for mesh datasets and (2) research work focused on developing new communication software named FAODEL that enables applications in a workflow to exchange data more efficiently.

IOSS: IOSS is a production I/O library for mesh datasets used in many Sandia applications. IOSS provides a common front end for mesh data and supports multiple back-end file databases (e.g., Exodus and CGNS). This project is enhancing IOSS is two ways. First, a new hybrid mesh capability is being developed that will allow IOSS to support both structured and unstructured grids. This capability will allow users to use one API to access both forms of geometry data, thereby allowing them to more easily choose a meshing strategy that matches an application's needs. Second, IOSS is being enhanced to natively support Burst Buffers. These extensions will simplify the process of using Burst Buffers and accelerate general storage performance.

FAODEL: One of the challenges for production simulation work in Exascale computing is that workflows are becoming increasingly more sophisticated. Analysts routinely use workflows that use a variety of parallel tools to groom datasets, simulate different effects, and analyze a collection of results. Coupling different simulation tools together into a monolithic job is challenging due to library incompatibilities and fundamental differences in application runtimes (e.g., bulk synchronous parallel vs. asynchronous many-task (AMT) programming models). Many workflows instead run each parallel tool as an independent job and use the file system as the mechanism for handing off data in the workflow.

This project is developing new communication software called FAODEL (Flexible, Asynchronous, Object Data-Exchange Libraries) [2] to provide better primitives for moving data between applications in Exascale platforms. In addition to supporting RDMA transfers of data from one MPI or AMT job to another, objects may be cached in distributed memory or nonvolatile memory resources.

Key Challenges The key challenges for IOSS improvements are largely related to production stability. Adding support for hybrid meshes and Burst Buffers must be done in a way that does not cause significant changes to current APIs, while at the same time exposes enough functionality to be useful.

The key challenges in developing FAODEL to provide new data management services include the following:

1. **Job-to-Job Support:** Workflows need a way to exchange data from one running application job to another in an efficient manner. While most RDMA communication libraries provide good *intra*-job performance, they do not provide mechanisms to support *inter*-job operations. Inter-job communication is complicated by the fact that the networks in many platforms have vendor-specific access controls that block inter-job communication by default. FAODEL must interface with these access controls. It must also avoid disturbing the application's native intra-job communication (e.g., MPI).
2. **Flexible Management of Resources:** Service developers need a flexible way to reason about how an application's in-memory datasets are exposed to other applications. In addition to hosting data in dedicated staging nodes, applications should be able to host data in place for efficiency.
3. **Asynchronous and Event-Driven Semantics:** AMT and analysis applications are asynchronous and require event-driven semantics that allow operations to trigger as objects migrate through the platform. Traditional remote procedure call (RPC) do not map well to this environment.

Solution Strategy FAODEL is designed to provide communication libraries to implement advanced data management services. As illustrated in Figure 51(a), it is composed of five core libraries: a low-level RDMA portability library (NNTI), an out-of-band RESTful API for establishing job-to-job communication (Webhook), a network memory manager (Lunasa), an asynchronous communication engine (OpBox), and a distributed key/blob service (Kelpie).

[2] In Gaelic, a *faodhail* is a land bridge that forms between islands at low tide.

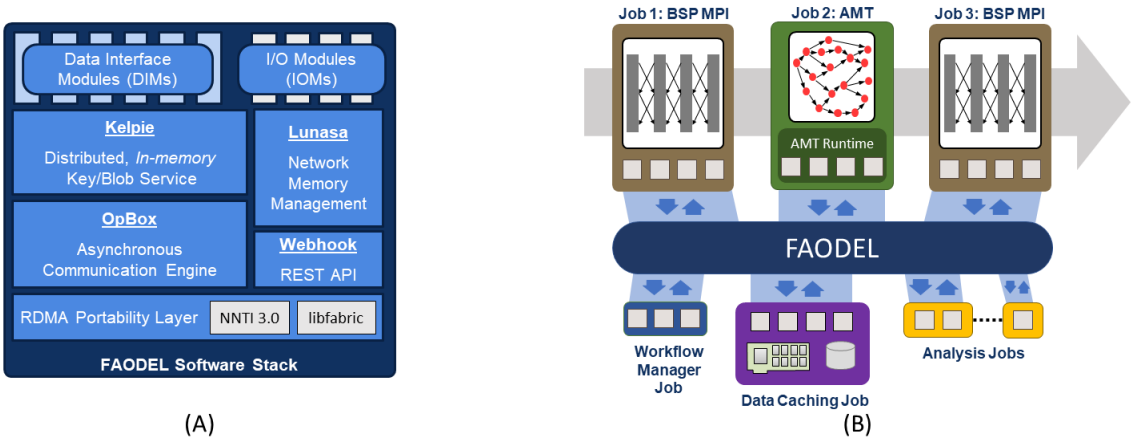


Figure 51: (a) FAODEL software stack and (b) a workflow example.

Figure 51(b) depicts an example of how multiple jobs of different types are connected together to implement a workflow. When a workflow manager dispatches these jobs to resources in the platform, it defines pools of nodes that are responsible for housing each dataset's objects. A pool may be internal or external to a job, and includes a definition of how data is distributed across pool members (e.g., a distributed hash table). Data transfers with a pool are coordinated through asynchronous communication state machines managed by OpBox. Downstream applications may choose to subscribe to data objects in a pool instead of polling for availability. If a requested object in a subscription is not available, OpBox pauses the request's state machine until it becomes available.

FAODEL users typically implement custom data interface modules (DIMs) to work with application-specific datasets. These interfaces decompose data into a collection of contiguous objects that can be dispersed to a pool as needed. While users are free to define how objects are structured and indexed in a DIM, the expectation is that data is organized in a form that is ready-to-use by applications. DIMs have been constructed for ATDM's EMPIRE and SPARC applications. A general mesh interface will be constructed to allow IOSS users to use FAODEL transparently.

Recent Progress

1. **FAODEL Integration into EMPIRE:** I/O interfaces have been constructed for EMPIRE/Fluid and EMPIRE/PIC to allow the applications to use FAODEL to store/retrieve mesh, field, and particle data. This implementation targets a checkpoint/restart use case but could be extended to couple EMPIRE to analysis tools.

2. **Improved Asynchronous Messaging:** FAODEL allows users to express a series of communication operations as a state machine that is updated as relevant events happen. FAODEL was updated to allow different state machines to be processed in parallel to improve concurrency.

3. **Initial Open Source Release:** FAODEL software was reviewed and approved for external release under the MIT license. The release is currently being transitioned to `https://github.com/faodel/faodel`.

Next Steps

1. **EMPIRE/FAODEL Performance Experiments:** The FAODEL interfaces for EMPIRE will be tested and benchmarked on the Trinity platform.

2. **Transition to Sierra:** FAODEL will be updated to run on the CORAL/Sierra architecture.

4.4.5 SNL ATDM Data and Visualization: Visualization Capabilities

Overview The SNL ATDM Data and Visualization work consolidates existing ATDM activities in scalable data management and visualization. Part of the responsibilities of the SNL ATDM Data and Visualization Project is the maintenance and development of visualization resources for ATDM applications on Exascale platforms. The ATDM Scalable Visualization project provides visualization and analysis required to satisfy the needs of the ASC/ATDM applications on next-generation, many-core platforms. This involves many activities including the re-engineering of visualization algorithms, services, and tools that enable ASC customers to carry out data analysis on ASC systems and ACES platforms. Current tools include scalable data analysis software released open source through ParaView [153], VTK [154], and Catalyst [155]. We are also both leveraging and contributing to VTK-m [156], a many-core visualization library, to satisfy our visualization needs on advanced architectures.

The scope of the Scalable Visualization under ATDM at SNL is R&D for the programming model and implementation of visualization code for ASC/ATDM projects and ASC/ATDM application support.

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 [153, 155]. 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.

Solution Strategy The SNL ATDM Visualization effort is addressing its challenges by leveraging and expanding the following technologies.

1. ***In situ* visualization** We are integrating our HPC visualization capabilities into our ATDM simulations by leveraging the Catalyst *in situ* visualization library [155]. Catalyst is part of the code supported by the ALPINE project (WBS 2.3.4.12), and we leverage this software by integrating it with ATDM application codes and applying it in our simulation runs.

2. **Data compression** Data compression allows us to make more effective use of the bandwidth and capacity of our storage systems. We are developing the TuckerMPI library [157] to provide compression specifically optimized for distributed parallel simulation data.

3. **Algorithm portability** Running visualization *in situ* with a simulation mandates executing the visualization algorithms on the same compute resources. Thus, our visualization software must be able to run on ASC systems and ACES platforms. We are leveraging the VTK-m software [156] being developed by the ECP/VTK-m project (WBS 2.3.4.13), optimizing the library for ACES platforms, and integrating the code with our ATDM software.

Recent Progress SNL ATDM has several ongoing visualization-centric activities.

***In Situ* Visualization** Under ATDM, several Catalyst adapters are under development for Sandia simulation codes. These adapters leverage a range of *in situ* technologies such as shallow copy of simulation data structures for reduced memory footprint, data compression of simulation grid data fields, and the use of VTK-m filters for on-node compute performance.

SPARC (Sandia Parallel Aero sciences Research Code) [158] is a massively parallel compressible CFD code that is designed to be highly scalable. SPARC uses a hybrid structured/unstructured mesh format, and there is ongoing development to construct a Catalyst adapter that efficiently uses memory to represent this data structure using shallow copy data structures provided by ParaView.

Nalu [159] is a massively parallel unstructured low Mach flow code. An adapter for Catalyst was developed for Nalu using the existing Seacas IOSS adapter for the Sierra framework to output unstructured mesh data to Catalyst. Examples of *in situ* visualization with Nalu are shown in Figure 52.

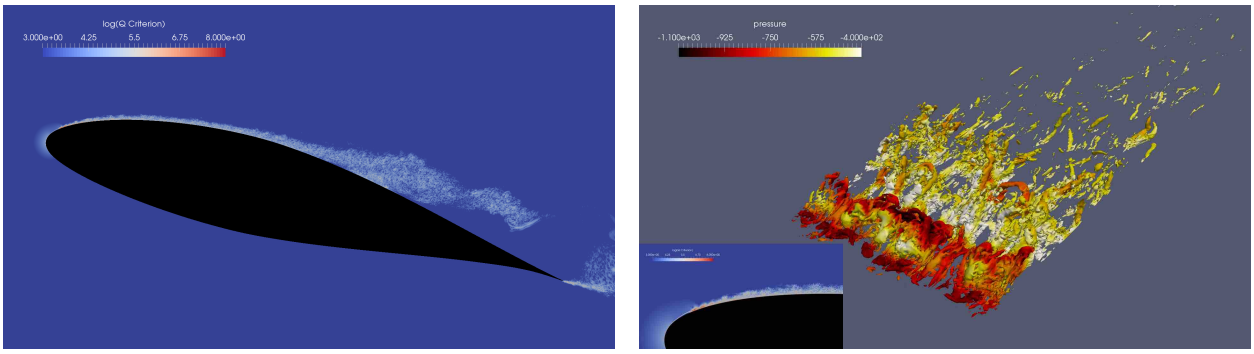


Figure 52: Examples of an *in situ* visualization of a Nalu simulation on 2560 processes of airflow over a wind turbine airfoil. At left: a cross-sectional slice through the airfoil along the direction of flow colored by Q criterion. At right: detail at the leading edge of the wind turbine airfoil.

Sparta [160] is a Sandia developed massively parallel DSMC (Direct Simulation Monte Carlo) code. Sparta uses a unique hierarchical Cartesian grid structure to track simulation particle movement and interaction. The Catalyst adapter implementation for Sparta uses shallow copy data structures to represent this hierarchy as an unstructured grid without forcing unnecessary copies of Sparta data structures.

Data Compression Although the TuckerMPI compression library is already available as an open standalone git repository [161], we pursued three independent approaches to integrate TuckerMPI compression with the *in situ* technologies being developed in parallel: (1) modify Catalyst adapter, integrated with a miniFE demo example, to perform TuckerMPI compression (copy input data, create tensor data structures, perform compression over distributed data), (2) create a standalone "TuckerWriter" plugin for ParaView, packaged as a standalone shared object library (links appropriately with the separately built Tucker Library) that can be loaded from a ParaView GUI, and, (3) extend the SPARC Catalyst direct output branch to work alongside SPARC writer output. This extension allows Catalyst output from SPARC input decks that request output in a specific file format (CGNS), and will be used to construct test cases for TuckerMPI compression.

In addition to TuckerMPI integration with ParaView/Catalyst, we are also conducting R&D to extend the capability to compress unstructured non-rectilinear mesh data using functional tensor approximations. We are researching an approach whereby multi-variate data on an unstructured mesh is treated as a high-dimensional function (spatial dimensions, time), which may be approximated as a small sum of product of univariate functions. Encoding the univariate functions requires storing only a small number of function coefficients/parameters, resulting in large compression.

Algorithm Portability The SNL ATDM project is helping VTK-m run better on ASC platforms with multiple activities. First, we are improving the parallel radix sorting operation in VTK-m. Parallel sorting is a core component of many of our visualization algorithms. Sorting helps us group common identifiers and conditions index sequences for fast searching. Improving the performance of sort within the VTK-m framework implicitly improves the performance of many algorithms that already use it. We are implementing a parallel radix sorting algorithm based on the work of Satish, et al. [162]. Our initial experiments indicate that the radix sort generally outperforms the default TBB sort algorithm typical data types.

Second, we are working to improve VTK-m's support vector instructions to improve performance on processors such as the Intel Xeon and Xeon Phi. Compilers work to identify code loops where vectorization can be applied, but in VTK-m changes were required to help the compiler. To enable vectorization across multiple complex data types, we are experimenting with loading values in blocks as we schedule an algorithm. This allows the compiler to better vectorize worklet functions that are now accessing well behaved blocks of memory. Given a sufficiently complex worklet, the benefit of better vectorization outweighs the overhead of maintaining blocks of values.

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

4.4.6 2.3.4.05 STDM07-VeloC: Very Low Overhead Transparent Multilevel Checkpoint/Restart

Overview The VeloC project aims to provide a 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 directly by the applications or indirectly by I/O libraries (e.g., HDF5, ADIOS, PnetCDF) to capture local application states, which are then coordinated and persisted using a resilience engine. The VeloC software intends to serve ECP applications such as HACC, NWChem, QMCPACK, LatticeQCD and EXAALT.

Key Challenges Applications typically employ simple checkpoint-restart mechanisms to survive failures that directly use a parallel file system. However, I/O bottlenecks to to concurrent writes are a known limitation in this case. At Exascale, failures are more frequent but the available file system I/O bandwidth per compute unit decreases. Therefore, there is a need to checkpoint more frequently but simple checkpointing becomes even less scalable and efficient. To compensate for the decreasing I/O bandwidth per compute unit, the storage stack is become increasingly more heterogeneous (e.g. NVRAM, burst buffers, key-value stores, etc.). This aspect introduces a new level of complexity for application developers: they need to adapt their checkpoint strategy to a variety of new storage sub-systems that may or may not be available on every machine. This is further amplified by the diversity of vendors that offer custom APIs. Thus, it is important to provide a scalable high-performance checkpointing solution that can leverage the heterogeneous storage stack transparently without sacrificing ease of use and flexibility.

Solution Strategy To address these challenges, VeloC is based on several design principles.

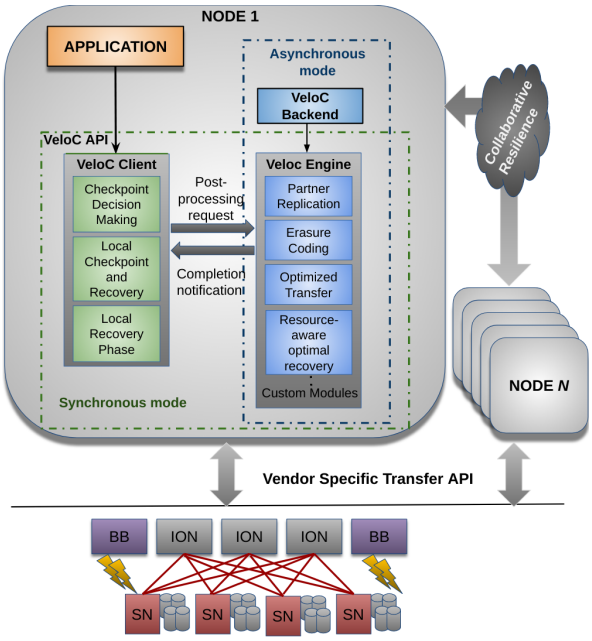
First, it implements multi-level checkpointing. Specifically, it persists the local checkpoints to other nodes using collaborative resilience strategies (e.g., partner replication and partner erasure coding) and to external storage that may include a complex heterogeneous hierarchy (e.g., parallel file system, burst buffers, I/O nodes, key-value stores, etc.). Based on experience from previous efforts such as FTI [163] and SCR [164], this strategy greatly improves performance and scalability.

Second, it offers a simple API that enables users to either manually capture checkpoints into node-local files or to define memory regions that are automatically serialized into node-local files. All interactions with the complex heterogeneous storage stack is transparently handled by VeloC, which facilitates ease-of-use.

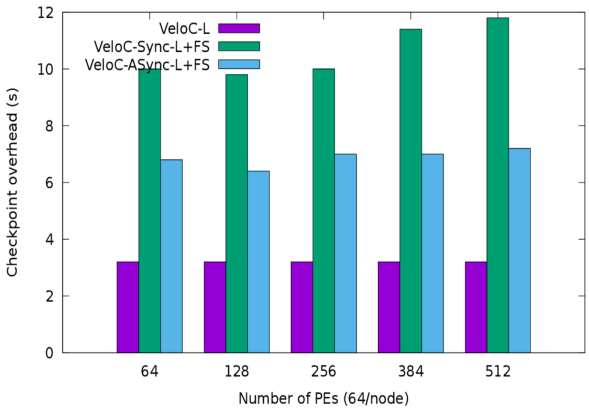
Third, VeloC separates the management of node-local checkpoints from the actual implementation of the resilience strategies by using a client library that interfaces with a resilience engine. The engine can be linked directly into the application and will run in synchronous mode, or it can be part of a separate backend process that facilitates an asynchronous mode where the resilience strategies are applied in the background to avoid blocking the application. This ultimately leads to better performance and scalability.

Recent Progress We have interviewed and met with several ECP application team in an effort to understand their checkpointing needs. Most of our current efforts involved the HACC, ECP NWChem-X, Lattice QCD and QMCPACK teams. These interviews helped us understand the needs to the ECP applications and ensure that they get reflected in the VeloC API specification. In the last several months, the VeloC team has developed a stable version of the VeloC API specification, which is currently in the process of being implemented.

Parallel to the VeloC API definition, we also worked on the architecture and design of the VeloC software. The VeloC software design is based on a modular and scalable architecture, as shown in Figure 53a. The design details of the VeloC software can be found in the VeloC design document, submitted to ECP during past milestones. Working off the design specification, we implemented the VeloC software according to the design principles introduced in the previous section. We have worked on understanding, refactoring, modularizing several components of the Fault Tolerance Interface (FTI) and Scalable Checkpoint/Restart (SCR) library to extract self-contained modules that can be used in the VeloC software. We recently completed the integration of self-contained modules and other driver modules into a flexible engine that allows VeloC the capability of running in synchronous mode directly in the application processes or in asynchronous mode in a separate active backend process.



(a) Architecture: modular client and engine (running in synchronous and asynchronous mode)



(b) Results: checkpointing overhead for a weak scaling experiment (local, local + sync mode, local + async mode). Lower is better

Figure 53: VeloC: Very Low Overhead Checkpoint-Restart

We have run an initial stress test of VeloC on the ANL Theta platform (KNL nodes, local SSD, Lustre parallel file system) using a heat distribution application benchmark. The test consists of a weak scaling experiment (64 processes per node) where the checkpointing overhead (runtime with a checkpoint vs. baseline without) is measured for three approaches: (1) node-local checkpoints written to the SSD; (2) node-local checkpoints followed by synchronous flush to the parallel file system; (3) node-local checkpoints followed by asynchronous flush to the parallel file system. The results are depicted in Figure 53b. As can be observed, asynchronous checkpoints are faster than synchronous checkpoints by a significant margin and more scalable.

We are also interacting with ALCF to understand the constraints for VeloC deployment on CORAL systems, such as the configuration of the resource and job manager and the possibility to run two MPI executions (application + back-end) on each node.

Next Steps The team will be working on the VeloC software release for the next quarter. Our goal is to finalize the integration and testing of all modules, clean up the code and run end-to-end testing. Furthermore, we will add documentation and tutorials to facilitate easy adopting in our user community.

4.4.7 ECP EZ: Fast, Effective, Parallel Error-bounded Exascale Lossy Compression for Scientific Data

Overview Extreme scale simulations and experiments are generating more data than can be stored, communicated and analyzed. Current lossless compression methods suffer from low compression ratio and do not adequately address the limitations in storage bandwidth and storage space of projected Exascale systems. Existing lossy compressors, although enabling greater data reduction, are not covering the needs of many ECP applications.

The EZ project is extending and improving the SZ lossy compressor for structured and unstructured scientific datasets respecting user-set error controls. SZ offers an excellent compression ratio 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 maximizing the effectiveness of SZ's three core compression algorithms: prediction, quantization, and coding. The EZ project focuses on optimization of the compression quality, including improvement of compression ratio, memory cost minimization, parallelization, addition of more error controls, and integration of SZ into parallel I/O environments (PnetCDF, ADIOS, HDF5). Our goal is to produce a high-quality lossy compressor responding to the needs of ECP Exascale applications and experiment users. To this end, we are working with multiple ECP application teams, including ExaSky cosmology teams (HACC and Nyx), molecular dynamics simulations groups (EXAALT, QMCPACK), x-ray laser imaging experimentalists (ExaFEL), and computational chemists (NWChem-X) to optimize SZ for their applications and to harden SZ for production use.

Key Challenges There are several key challenges in the EZ project.

- One key challenge in designing an efficient compressor for HPC applications is the large diversity of scientific simulation data, such as various dimensions, different scales, and dynamic data changes in both space and time.
- Another challenge is that the scientific data may have irregular characteristics. The simulation data may exhibit spiky changes in small local areas. For instance, all the key data in the HACC code are stored in six 1D arrays: three coordinate fields (xx, yy, zz) and three velocity fields (vx, vy, vz). It is hard to get a high compression ratio because of lack of correlations between adjacent particles in each array.
- It is non-trivial to parallelize the lossy compressor SZ to get a high speed-up because of the relatively strong data dependencies in the data compression and decompression phase. The data prediction step, for example, relies on the neighboring data points for each data point, such that the entire compression/decompression has to be split into multiple blocks. How to perform the block-wise compression efficiently is an uneasy issue because block-wise prediction may degrade the prediction accuracy.
- Integrating SZ into parallel I/O libraries (such as PnetCDF and HDF5) is non-trivial, because these I/O libraries are designed and implemented differently with various interfaces. We need to develop the integration codes based on the I/O libraries' interface with minimal cost.

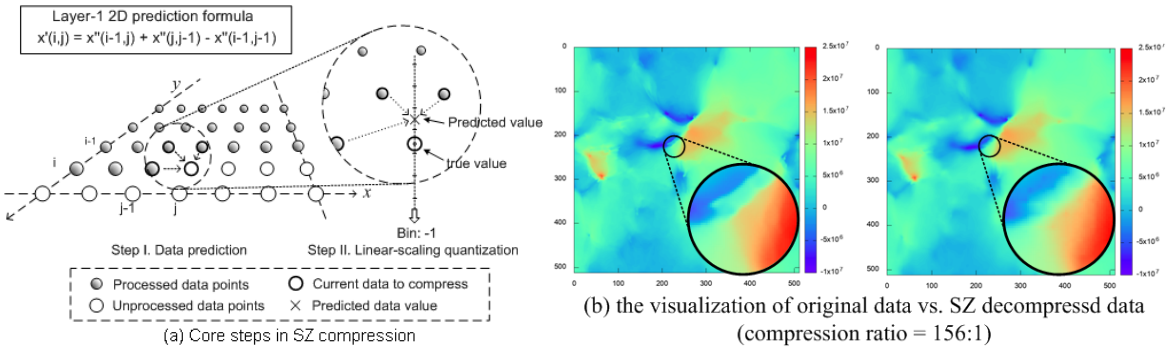
Solution Strategy As for the first challenge, we keep a close communication with ECP application users to understand their specific demands on the lossy compression. For instance, we have a weekly meeting with ExaSky team to discuss the required error bounds and compression quality of cosmology simulation data. We also provide multiple types of error bounds (such as absolute error bound, PSNR, relative error bound) allowing users to control the errors in different ways.

As for the second challenge, we develop some particular compression techniques based on specific data features across different applications.

As for the third challenge, we split the entire dataset into multiple blocks based on the number of threads/ranks and conduct an adaptive prediction method to optimize the prediction accuracy dynamically.

We overcome the last challenge by understanding the principle of the I/O libraries (either reading the related documents or communicating with the I/O library developers closely).

Recent Progress SZ has been released as an open source on GitHub (http://github.com/disheng222/SZ). Figure 54 (a) demonstrates the core step (data prediction and linear-scaling quantization) in SZ lossy compression using a 2D dataset. Our customized improved compression techniques for different datasets mainly focus on various prediction methods based on data features. Figure 54 (b) demonstrates the visual quality of decompressed data for ExaSky-NYX VX field. We can see that SZ has a pretty high visual quality even with a high compression ratio up to 156:1.



(a) Core steps in SZ compression

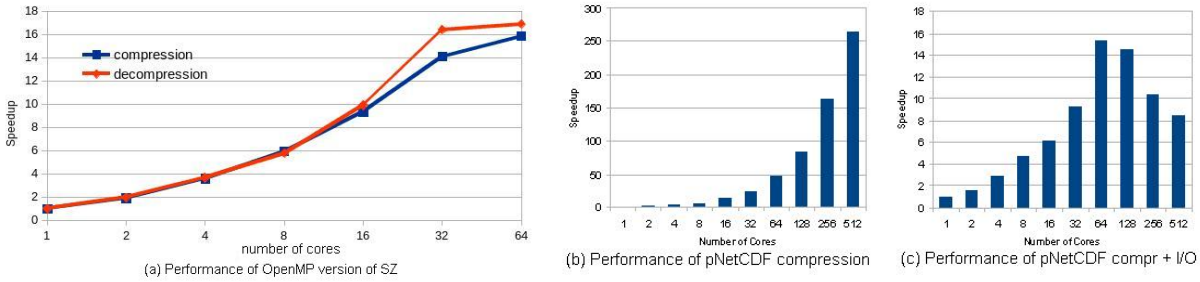
(b) the visualization of original data vs. SZ decompressd data (compression ratio = 156:1)

Figure 54: Illustration of data prediction in SZ and visual quality of decompressed data for ExaSky-NYX VX field

We have improved the compression quality for some ECP applications significantly. For instance, we develop PaSTRI code that can significantly improve the compression ratio for GAMESS two-electron integral datasets by leveraging the potential scaled repeated patterns. We also implemented effective compression method supporting point-wise relative error bounds for the ExaSky project and studied its lossy compression quality with multiple resolutions. Experiments with point-wise relative error bound based compression shows that our solution leads to 31%-210% higher compression ratio than other lossy compressors do.

We developed the multi-thread OpenMP version for SZ and evaluate the parallel compression performance, as presented in Figure 55 (a). It is observed that the compression can obtain a speed-up of 250+ and the overall performance using lossy compression+I/O can be improved 15X compared with the data I/O without lossy compressor. we can observe that the OpenMP version of SZ obtains about 6:1∼16:1 performance gain when using 8∼64 threads to do the compression/decompression in parallel. Decompression has higher performance gain than compression as the number of cores grow because it does not need to construct the Huffman tree.

We completed the integration of SZ into three popular I/O libraries (PnetCDF, ADIOS, and HDF5). Figure 55 (b) and (c) present the performance evaluation using PnetCDF integrated with SZ.



(a) Performance of OpenMP version of SZ

(b) Performance of pNetCDF compression

(c) Performance of pNetCDF compr + I/O

Figure 55: Performance evaluation of SZ with OpenMP and PnetCDF

Next Steps Our next efforts are:

Improve compression ratio by leveraging the smoothness of the data in the time dimension. (1) Explore techniques for compression in time dimension; (2) Integration of compression in time dimension. (3) Test and performance evaluation on available CORAL systems

4.4.8 UnifyCR – A file system for burst buffers

Overview The view of storage systems for HPC is changing rapidly. Traditional, single-target parallel file systems have reached their cost-effective scaling limit. As a result, hierarchical storage systems are being designed and installed for our nation's next-generation leadership class systems. Current designs employ "burst buffers" as a fast cache between compute nodes and the parallel file system for data needed by running jobs and workflows. Burst buffers are implemented as compute-node local storage (e.g., SSD) or as shared intermediate storage (e.g., SSD on shared burst buffer nodes).

Because burst buffers present an additional complexity to effectively using supercomputers, we are developing UnifyCR, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. UnifyCR will address a major usability factor of the CORAL and other future systems, because it will enable applications to gain the performance advantages from distributed burst buffers while providing ease of use similar to that of a parallel file system.

Key Challenges The hierarchical storage for future HPC systems will include compute-node local SSDs as burst buffers. This distributed burst buffer design promises fast, scalable I/O performance because burst buffer bandwidth and capacity will automatically scale with the compute resources used by jobs and workflows. However, a major concern for this distributed design is how to present the disjoint storage devices as a single storage location to applications that use shared files. The primary issue is that when concurrent processes on different compute nodes perform I/O operations, e.g., writes, to a shared file, the data for the file are scattered across the separate compute-node local burst buffers instead of being stored in a single location. Consequently, if a process wants to access bytes from the shared file that exist in the burst buffer of a different compute node, that process needs to somehow track or look up the information for locating and retrieving those bytes. Additionally, there is no common interface across vendors for accessing remote burst buffers, so code for cross-node file sharing will not be easily portable across multiple DOE systems with different burst buffer architectures, further increasing programming complexity to support shared files.

For the reasons outlined above, it is clear that without software support for distributed burst buffers, applications will have major difficulties utilizing these resources.

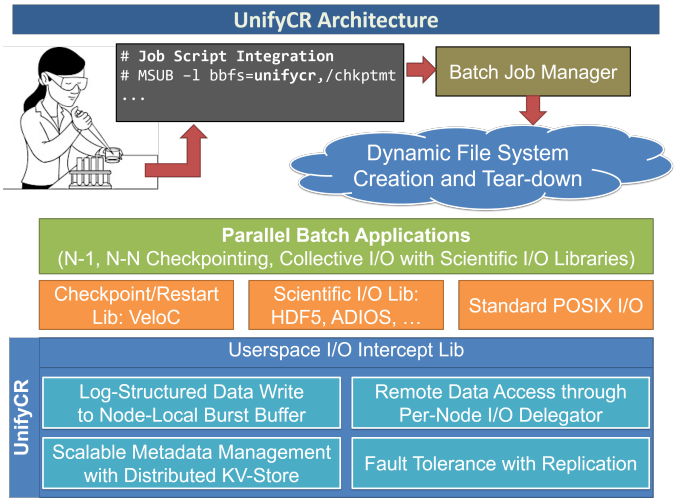


Figure 56: UnifyCR Overview. Users will be able to give commands in their batch scripts to launch UnifyCR within their allocation. UnifyCR will work with POSIX I/O, common I/O libraries, and VeloC. Once file operations are transparently intercepted by UnifyCR, they will be handled with specialized optimizations to ensure high performance.

Solution Strategy To address this concern, we are developing UnifyCR, a user-level file system, highly-specialized for shared file access on HPC systems with distributed burst buffers. In Figure 56, we show a

high level schematic of how UnifyCR will work. Users will load UnifyCR into their jobs from their batch scripts. Once UnifyCR 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 UnifyCR mount point will be intercepted and handled with specialized optimizations that will deliver high I/O performance.

Because checkpoint/restart has been reported to cause 75-80% of the I/O traffic on some HPC systems, we target our approach at checkpoint/restart workloads. Thus, UnifyCR will address a major usability factor of the CORAL and other future systems. We will design UnifyCR such that it transparently intercepts I/O calls, so we expect it to integrate cleanly with other software including I/O and checkpoint/restart libraries. Additionally, because UnifyCR is tailored for HPC systems and workloads, it can deliver high performance.

Recent Progress In the first year of this project, the team spent the bulk of the time evaluating the research prototype code that forms the basis for UnifyCR and designing the framework for the production version of UnifyCR (See Figure 57). The major changes we plan to make to the research prototype code include: replacing MPI communication with DataLib software; integrating UnifyCR with resource managers; implementing support for advanced storage devices (burst buffers); and implementing support for I/O libraries and VeloC.

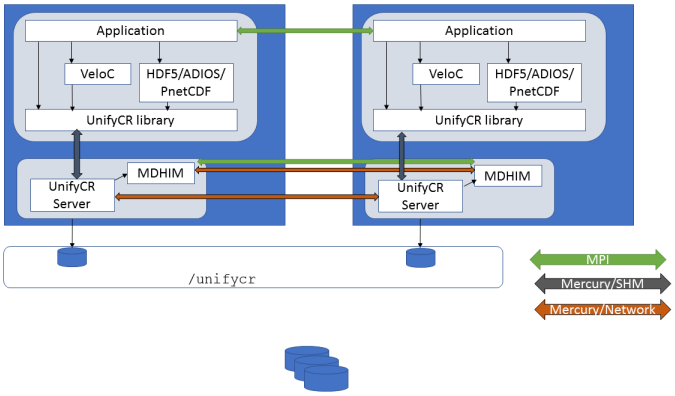


Figure 57: UnifyCR Design. The UnifyCR instance will consist of a dynamic library and a UnifyCR daemon that runs on each compute node in the job. The library will intercept I/O calls to the UnifyCR mount point from applications, I/O libraries, or VeloC and communicate them to the UnifyCR daemon that will handle the I/O operation.

Upon completing the evaluation and design milestones, we released an alpha version of the code UnifyCR 0.1 on GitHub at https://github.com/LLNL/UnifyCR. Subsequently, we implemented support for easily starting and stopping UnifyCR from within a user batch script. In addition, we have performed other code changes to ensure production ready use of UnifyCR, including adding testing, improving the build mechanisms, improving the I/O interception functionality; and including code documentation.

Next Steps For our next milestone effort, we are going to be implementing support for I/O libraries into UnifyCR. We are targeting HDF5, MPI-IO, PnetCDF, and ADIOS. When user applications utilize these high level I/O libraries, we will ensure that their I/O operations are intercepted and handled as easily as if they came from POSIX in the application.

Following this milestone, we will begin the effort of supporting the advanced storage hardware that will be available on the upcoming CORAL systems at LLNL and ORNL (Sierra and Summit).

4.4.9 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, etc. 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, various HDF5 features are in development to address efficiency and other challenges posed by data management and parallel I/O on Exascale architectures. The ExaHDF5 team is productizing features and techniques that have been previously prototyped, exploring optimization strategies on upcoming architectures, maintaining and optimizing existing HDF5 features tailored for ECP applications. Along with supporting and optimizing 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 and multi-reader (SWMR) for workflows, asynchronous I/O, data and metadata querying.

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 HDF5. Full SWMR feature will support the needs of ExaFEL's workflow in appending data incrementally. Virtual Object Layer (VOL) and interoperability features with netCDF and ADIOS data opens up the rich HDF5 data management interface to a large number of 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 industry 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. In addition, support for higher bandwidth transfers and lower message latency, interconnects in supercomputers are becoming more complex, in terms of both topology 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 effectively.

Interoperability with other file formats. HDF5 offers a rich data model and strong features for accessing data, and using these capabilities to access data stored in other data formats would be a valuable productivity boost to applications. The team is developing to add interoperability features that will enable ECP applications to use HDF5 function calls to directly read from other file formats. Development includes functionality to read "classic" netCDF (including PnetCDF) and ADIOS/BP files, as these formats are in active use in DOE application communities focused on Exascale deliverables.

Solution Strategy *Utilizing complex compute and storage hardware.* To use multi-level storage hierarchy, Data Elevator is being developed in this project. The Data Elevator library intercepts HDF5 file open calls and redirects them to intermediate and faster caching storage layers, where application reads or writes data. When updates or writes to the intermediate data is done, a server daemon of the Data Elevator moves the data transparently to its final destination on colder storage layers, such as disk-based parallel file system, without

modifying the source code or placing the burden on user to move the data explicitly from the intermediate storage layer.

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

Interoperability with other file formats. To open the HDF5 API for interfacing with various file formats and to provide capability of intercepting HDF5 API calls, Virtual Object Layer (VOL) feature is being developed. the Virtual Object Layer (VOL) adds a new abstraction layer internally to the HDF5 library and is implemented just below the public API. The VOL intercepts all HDF5 API calls that access objects in a file and forwards those calls to an "object driver" plugin. A VOL plugin can store HDF5 data model objects in a variety of ways. Figure 58 shows a high-level view of VOL, where intercepted HDF5 API can interface with other file formats and object storage. We will create VOL plugins to access the netCDF and ADIOS/BP file formats, so that applications can use the HDF5 API to operate on data stored in these formats. The interoperability functions in the VOL support the pre-defined datatypes (integers, floating-point values, strings, etc.) in these formats, and will also support compound datatypes, i.e., a user-defined combination of pre-defined datatypes, in netCDF and will use compound datatypes to support ADIOS' complex datatypes, which represent complex numbers.

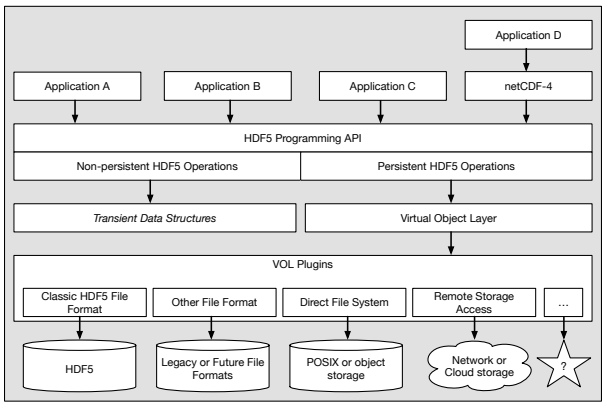


Figure 58: An overview of Virtual Object Layer (VOL)

Recent Progress *Prototype implementation of Data Elevator.* The project team has developed a prototype implementation of write caching functionality and deployed it on the Cori system at NERSC. Using the burst buffers on Cori, the Data Elevator achieves 4x to 6X performance improvement over a highly tuned HDF5 code in writing large data. Performance evaluation included representative I/O kernels from Chombo, which is an AMR library supporting subsurface simulation, and from a space weather simulation that operates on billions to trillions of particles.

Syncing VOL branch with the develop branch. VOL branch has been brought in sync with the latest development branch. The previous VOL prototype branch was developed 3 years ago and the development branch has evolved significantly since then.

Supporting ECP application I/O The ExaHDF5 team has been working with various applications in the ECP portfolio. Teams we have been working 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, etc. By simply changing these configurations, HDF5 achieved higher performance in writing files. A few minor bugs also have been identified and fixed to improve metadata read/write performance.

Next Steps *Release of VOL in HDF5.* The VOL integration branch is being merged with the development branch. Once all the testing is complete and regression testing for maintenance is in place, the VOL feature will be released as part of the main HDF5 software.

Release of Data Elevator write caching. Testing and evaluation of Data Elevator write caching is in progress. After rigorous testing is complete, the feature will be released as a VOL plugin. It will also be installed on the Cori system at NERSC and any other systems that have burst buffers at the edge of compute nodes. Development of read caching and prefetching, and use of node-local burst buffers is in progress.

Release of data write caching with Data Elevator.

4.4.10 ADIOS

Overview The Adaptable I/O Systems, ADIOS [165], 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).

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 In the first year of this project, the ADIOS team has designed the new version of Application Programming Interface that unifies staging I/O and file I/O [166], and designed and created the

new object-oriented code framework [167] with basic functionalities, including writing and reading files in two different file formats (ADIOS BP format and HDF5 format). The new framework focuses on sustainable development and code reusability, which will help in the long term to add more and more new features to ADIOS. In the decade-long history of ADIOS many research artifacts has been incorporated and this tradition is expected to continue. The team has also designed the new scalable staging transport learning from the many lessons from using ADIOS for data staging and code coupling by applications in the past. As can be seen in Figure 59, this past experience with methods and deep science engagements has led to demonstrations at Titan scale.

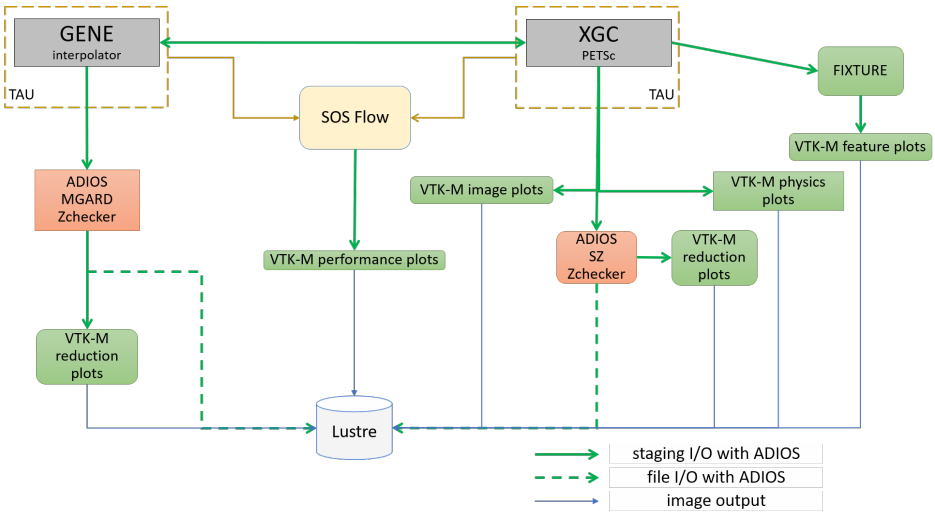


Figure 59: An example of using ADIOS to support ECP science. This sketch represents the demonstration at the February 2018 ECP Meeting, which featured WDM Fusion, CODAR, ADIOS, and other joint ECP activities. Note that all of the green arrows in the figure represent data communication or storage handled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

The new design focuses on stability and scalability so that applications can rely on it in daily production runs just as they have relied on the high performance file I/O of ADIOS. The new code base is governed with state-of-the art software development practices, including GitHub workflow of Pull-Requests with reviews, continuous integration that enforces well-tested changes to the code only, and nightly builds to catch errors on various combinations of architecture and software stack as soon as possible. The team has access to and the code is continuously tested on DOE machines (Titan, Summit-dev, Cori and Theta).

Next Steps In the second year, the team aims to transfer most functionality from the ADIOS 1.x software into the new code base. The goal is to reach parity in performance and functionality while succeeding in stability and code quality. Static and dynamic analysis will be integrated to the GitHub workflow to catch errors before they cause trouble. Code coverage tools will help with increasing code quality. Large effort is going into the development of the scalable staging transport and will start using it in in situ analysis, code coupling and in situ visualization in ECP applications to explore and evaluate its usability, scalability and stability.

4.4.11 DataLib

Overview Data Libraries and Services Enabling Exascale Science (DataLib). The DataLib project encompasses multiple related activities in user-level storage and I/O support for ECP codes on upcoming DOE platforms, providing a number of options for effectively storing and retrieving data and tools for assessing I/O behavior and performance. 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 **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 **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).

Key Challenges Each of these subprojects has its own set of challenges. Libraries such as 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 *Transparently refactoring application I/O.* Collective I/O is an important optimization in codes where phases of I/O are naturally identifiable: individual I/O operations from application processes are transparently refactored (i.e., without application code changes) to achieve the same goals at lower cost. We will improve collective I/O techniques for Exascale through tighter integration with I/O forwarding layers, topology awareness, and employing collective approaches in higher-level libraries where additional information is available.

Intermediate storage and alternative data organizations. Nonvolatile memory (NVM) or solid-state disk (SSD) integrated into platforms provides a pool of fast storage where data can be stored temporarily or staged for writing to an external store (e.g., parallel file system). Recognizing that this data is transient, alternative organizations (e.g., provide higher performance.

Capturing and reconstructing I/O behavior. Darshan characterizes fully the POSIX and MPI-IO layers and captures some statistics for HDF5 and PnetCDF. We will improve our coverage of HDF5, PnetCDF, and ADIOS to better enable understanding of these codes. We will reach out to other library teams to consider characterization of them as well (e.g., SCR, FTI, PIO, SILO).

Mercury porting and support for relevant platforms. Mercury is an RPC/bulk data communication library for use in HPC services and co-developed by the HDF Group. We will work with system vendors to enable efficient Mercury communication on the platforms of interest (e.g., UCX/PAMI on IBM).

Data service co-design with application teams. A number of candidate ECP applications have data service needs (e.g., material property databases, producer-consumer data pipelines, multi-scale/physics simulation coupling, multi-modal data organization and indexing). We will identify application teams with data service requirements that are particularly ill-suited to solution by vendor offerings (e.g., parallel file systems, standard

burst buffers, cloud-based databases) and co-design, develop, and evolve specialized services based on our components to meet their specific requirements for ECP.

Recent Progress *Darshan.* We added Python bindings to the log parsing library to facilitate analysis and created a module for Autoperf (https://www.alcf.anl.gov/user-guides/automatic-performance-collection-autoperf), used in ALCF for performance counter and MPI information. An initial version of HXHIM module for Darshan has been completed (towards STDM12-8). We are iterating with HXHIM development team on what more interesting statistics could be collected on the client side (where Darshan runs). This interaction has also accelerated work with Thallium by the HXHIM team, a layer for enabling use of Mercury. We released Darshan 3.1.6 with bug fixes for specific application problems.

Mochi and Mercury. Nightly tests are now running on Aires, InfiniBand, and OmniPath networks (STDM12-7). See below for more information on STDM12-7.

PnetCDF. We are revising APIs to reduce memory footprint (towards STDM12-10), migrating PnetCDF wiki pages to GitHub (https://parallel-netcdf.GitHubio), and working on driver to enable netCDF-4 access from PnetCDF (towards SDTM12-11). We are also adjusting the API in PnetCDF to better support I/O patterns with very large numbers of small requests, something we are seeing in E3SM workloads (via their PIO library layer). Finally, we are evaluating burst buffer features of PnetCDF on Theta, using local SSDs.

Figure 60 illustrates a refactoring of the PnetCDF library to include a dispatch layer. The dispatch layer provides a mechanism for implementing multiple back-ends under the standard API. This functionality is being used to enable writing in a log format for intermediate data and access of data in the netCDF-4 format (stored in HDF5 files).

ROMIO. We implemented a bug fix for single MPI-IO requests of larger than 4GiB on some platforms, added a check for monotonicity of offsets in types, an MPI requirement, and tested logFS backend on Cori. We have begun integration of pipelined I/O functionality into ROMIO (STDM12-9), specifically into the Lustre driver. We are also debugging the LogFS ROMIO back-end.

ParSplice. We are pushing KV updates back into Mochi repository, refining Spack build process, and working through bugs with ParSplice use of Mochi components in newest revisions.

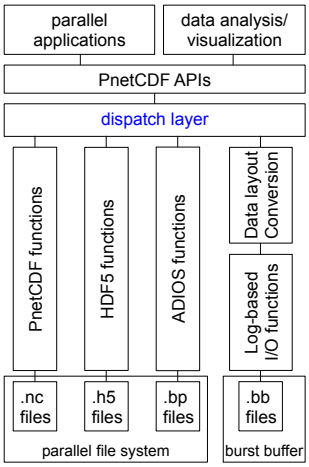


Figure 60: The new PnetCDF dispatch layer provides flexibility to target different back-end formats and devices under the PnetCDF API used by many existing applications.

Next Steps Next we will demonstrate a non-POSIX I/O characterization module for representative ECP I/O library (STDM12-8), demonstrate a PnetCDF prototype burst buffer backend on relevant ECP platform (STDM12-10), and release ROMIO including a pipelined I/O capability (STDM12-9).

4.4.12 ZFP: Compressed Floating-Point Arrays

Overview One of the primary challenges for Exascale computing is overcoming the performance cost of data movement. Through simulation, observation, and experiments, far more data is being generated than can reasonably be stored to disk and later analyzed without any 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, allowing for instance 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 are being developed that primarily exploit fine-grained data parallelism, but which also take advantage of task parallelism.
- **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 This project has made progress on several fronts over the past year to make the ZFP software [168] more Exascale ready and capable. Improved software development practices have been adopted, such as continuous integration, unit testing, extensive documentation [169], and portable build processes based on CMake and SPACK. A novel C equivalent of name spaces has been developed to allow linking to multiple versions of ZFP in the same library or executable to enable access to persistent files written using an evolving ZFP CODEC. ZFP compressed arrays can now be accessed using primitives with semantics equivalent to those commonly available for uncompressed scalar types and containers, such as proxy references and pointers, iterators, and flat and nested views and slices. The work on views has also laid the ground for enabling thread-safe access to ZFP compressed arrays, e.g., by enabling shared, read-only access among threads, in addition to partitioning into independent sub-arrays for read-write multithreaded access. ZFP has further been parallelized to support both OpenMP and CUDA execution, yielding throughput as high as 20 GB/s using 32 threads.

Finally, significant effort has been made to deploy ZFP to ECP tools and applications. For instance, ZFP compression is available in the ADIOS and HDF5 I/O libraries and has been specialized for the Cinema database format for in-situ visualization. A compressed format is being developed for the CEED high-order finite-element co-design center. Compression studies have been performed for the QMCPACK, ExaSMR, and ExaSky ECP applications, in which ZFP compressed arrays are being considered.

The results of these R&D efforts have been documented through publications [170, 171], 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 [172, 173, 174].

Next Steps Efforts are underway to provide C, Python, and Fortran bindings for ZFP arrays to facilitate integration with applications not written in C++. Additionally, new capabilities are being added to the main compression CODEC to support 4D and higher-dimensional arrays; missing/undefined array values and special values like NaNs and infinities; and lossless compression. Our CUDA implementation will serve as the basis for a new GPU-based compressed array primitive for VTK-m. Longer-term efforts include strategies for coping with unstructured data and spatially adaptive compressed arrays.

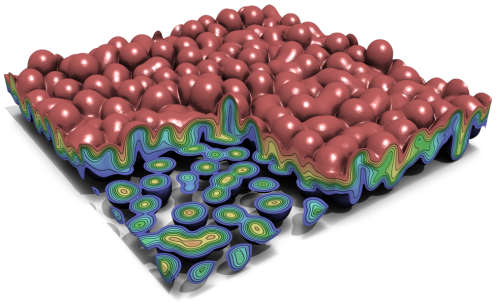


Figure 61: 240:1 ZFP compressed density field.

4.4.13 ALPINE

Overview ECP ALPINE will deliver in situ visualization and analysis infrastructure to ECP Applications. ALPINE developers come from the ParaView [175, 176] and VisIt [177] teams and ALPINE solutions will deliver in situ functionality in those tools, as well as ASCENT [178], a new in situ solution that focuses on flyweight processing. The ALPINE team 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, meaning they write data to disk and then visualize and analyze it afterwards. Given Exascale I/O 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 in situ-appropriate algorithms. For delivering infrastructure, the challenge is that our existing tools are not ready for Exascale In particular, we are concerned about achieving performance within simulation codes' time budgets, supporting many-core architectures, scaling to massive concurrency, and leveraging deep memory hierarchies. For in situ-appropriate algorithms, the challenge is that our stakeholders need to be able to apply in situ processing effectively without a human being in the loop. This means that we must have approaches to automate saving either the correct visualizations or the correct data extracts.

Solution Strategy A major strategy for our team is to leverage existing, successful software, ParaView and VisIt, including their recent developed in situ libraries Catalyst [155] and Libsim [179], and then to integrate and augment them with ALPINE capabilities to address the challenges of Exascale. Both software projects represent long-term DOE investments, and they are the two dominant software packages for large-scale visualization and analysis within the DOE Office of Science (SC) and the DOE National Nuclear Security Agency (NNSA). These two products will provide significant coverage of ECP Applications, and we can leverage their existing engagements to deliver ALPINE's algorithms and infrastructure. We are also developing another in situ library, ASCENT, with also utilizes this code repository; ASCENT is a "flyweight" solution, meaning that it is focused on a streamlined API, minimal memory footprint, and small binary size.

In terms of specifics, our solution strategy is two-fold, in response to our two major challenges (infrastructure and algorithms).

For infrastructure, we have developed a layer on top of the VTK-m library for ALPINE algorithms. This layer is where all ALPINE algorithms will be implemented, and it is deployed in ParaView, VisIt, and ASCENT. This means that all development effort by ALPINE will be available in all of our tools. Further, by leveraging VTK-m, we will be addressing issues with many-core architectures. Figure 62 illustrates our software strategy.

For automating in situ, we are pursuing four different algorithms:

- **Feature-based analysis** with moments to detect rotation-invariant patterns. These patterns can then be used either to direct which features should be visualized or to direct which extracts should be saved.
- **Lagrangian analysis** of vector flow allows more efficient and complete analysis and tracking of flow. It is a method for saving vector field data with more higher accuracy and less storage than the traditional approach.
- **Topological analysis** can be used to detect features in the data and steer visualizations. For example, contour trees can identify the most significant isosurfaces in complex simulations and then the resulting visualizations can use these isosurfaces.

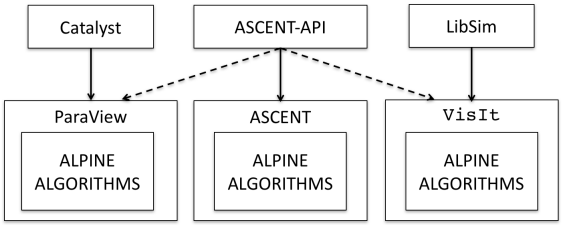


Figure 62: ALPINE's strategy for delivering and developing software. We are making use of existing software (ParaView, VisIt), but making sure all new development is shared in all of our tools. The dotted lines represent future work, specifically that the ASCENT API will work with ParaView and VisIt.

- **Importance sampling** can be used to guide visualizations and extracts to the most important parts of the simulation. Examples ranges from clustering similar data points within a region to identifying important time slices to save.

Recent Progress Again, we organize our recent progress around infrastructure and algorithms.

On the infrastructure side, we have completed the layer on top of VTK-m for ALPINE algorithms (Milestone STDA04-2, https://gitlab.kitware.com/vtk/vtk-m). We have stood up ASCENT, our flyweight in situ library, including defining its API (STDA04-1), making an initial release (STDA04-11), and making a production release (STDA04-30, https://github.com/Alpine-DAV/ascent).

On the algorithms side, we have been researching and developing our four algorithms (STDA04-5). In all cases, the progress is a combination of evaluating best approaches, and developing VTK-m based solutions that can be deployed in our infrastructures. Our topological analysis work is likely the furthest along. With this work, there is now a VTK-m implementation, although it needs additional effort for distributed memory support. The algorithm was applied to a WarpX simulation (STDA04-15), and used to automatically select isovalues in situ. Figure 63 compares the traditional approach with our enhanced version.

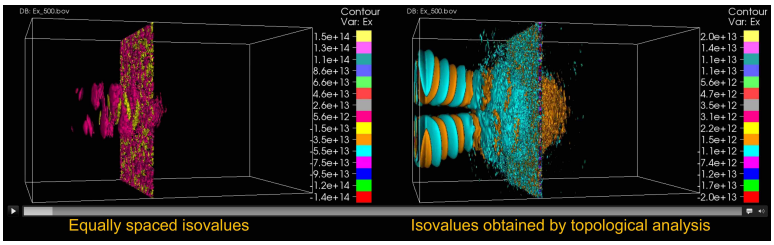


Figure 63: In situ visualizations taken from WarpX using VisIt. At left, equally spaced isovalues in an ion accelerator simulation. At right, our method chooses isovalues using topological analysis to more fully represent complex behavior in the data.

Next Steps

- For algorithm development, our next major activity is to develop scalable in situ versions of our algorithms (STDA04-31).
- For infrastructure, our next major activities are to integrate our algorithms into our infrastructures (STDA04-32) and deliver a release (STDA04-33).
- For ECP Application engagement, our strategy is to engage with Co-Design centers and connect with ECP applications through the centers. We have been partnering with AMReX and are integrating our in situ infrastructures into this package. We also have been engaging with the simulation teams that use AMReX.

4.4.14 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 and many integrated core. 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 [180, 181]. However, this decomposition works best when each processing element has on the order of a hundred thousand to a million data cells [182] and is known to break down as we approach the level of concurrency needed on modern accelerators [183, 184].

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 [156] 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 [185].

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 [186]. It has been shown that this approach not only simplifies parallel implementations, but also allows them to work well across many platforms [187, 188, 189].

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 [190] and array wrappers that can adapt to target memory layouts [191].

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.

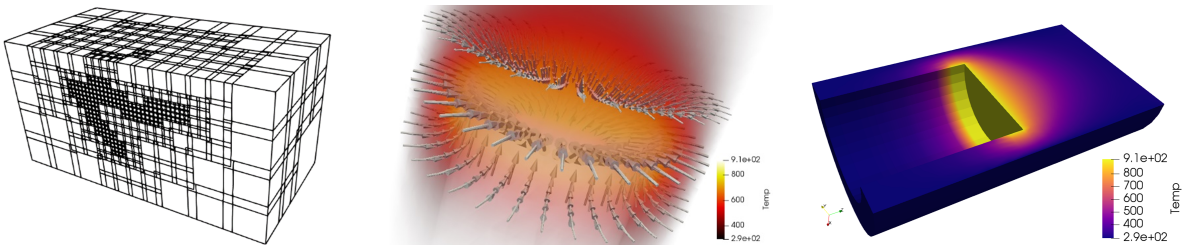


Figure 64: Examples of recent progress in VTK-m include (from left to right) multiblock data structures, gradient estimation, and mapping of fields to colors.

Recent Progress The VTK-m project is organized into many implementation activities. The following features have been completed in the past 12 months.

- **Key Reduce Worklet:** This adds a basic building block to VTK-m that is very useful in constructing algorithms that manipulate or generate topology [192].
- **Spatial Division:** Introductory algorithms to divide space based on the distribution of geometry within it. This is an important step in building spatial lookup structures.
- **Basic Particle Advection:** Particle advection traces the path of particles in a vector field. This tracing is fundamental for many flow visualization techniques. Our initial implementation works on simple structures
- **Surface Normals:** Normals, unit vectors that point perpendicular to a surface, are important to provide shading of 3D surfaces while rendering. These often need to be derived from the geometry itself.
- **Multiblock Data:** Treat multiple blocks of data, such as those depicted in Figure 64 at left, as first-class data sets. Direct support of multiblock data not only provides programming convenience but also allows us to improve scheduling tasks for smaller groups of data.
- **Gradients:** Gradients, depicted in Figure 64 at center, are an important metric of fields and must often be derived using topological data. Gradients are also fundamental in finding important vector field qualities like divergence, vorticity, and q-criterion.
- **Field to Colors:** Pseudocoloring, demonstrated in Figure 64 at right, is a fundamental feature of scientific visualization, and it depends on a good mechanism of converting field data to colors.
- **VTK-m 1.1 Release:** VTK-m 1.1 was released in December 2017.

Next Steps Our next efforts include:

- **External Surface:** Extracting the external faces of solid geometry is important for efficient solid rendering.
- **Location Structures:** Many scientific visualization algorithms require finding points or cells based on a world location.
- **Dynamic Types:** The initial implementation of VTK-m used templating to adjust to different data structures. However, when data types are not known at compile time, which is common in applications like ParaView and VisIt, templating for all possible combinations becomes infeasible. Provide mechanisms to enable runtime polymorphism.
- **OpenMP:** Our current multicore implementation uses TBB [193] for its multicore support. However, much of the code we wish to integrate with uses OpenMP [194], and the two threading implementations can conflict with each other. Thus, add a device adapter to VTK-m that uses OpenMP so this conflict will not happen.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND 2018-4009 R

4.5 SW ECOSYSTEM & DELIVERY

This section present projects in SW Ecosystem & Delivery.

4.5.1 Software Development Kits

Overview The ST Software Development Kit (SDK) project supports a set of activities aimed at

- establishing Community Policies aimed at increasing the interoperability between and sustainability of ST software packages, using the xSDK community package [195] and installation [196] policies as a model (reference to xSDK section?).
- coordinating the delivery of a comprehensive and coherent set of software tools to all interested stakeholders on behalf of ECP ST. This includes ECP applications and the broader open source community.

SDK 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 on facility platforms. This project will work closely with the HI 2.4.4 *Deployment of Software at the Facilities* project.

Key Challenges ST software packages have been developed in a variety of very different cultures and are at significantly different levels of software engineering maturity and sophistication. The experience of some of the SDK staff during the formation of the xSDK showed that in this situation, it is challenging to establish common terminology and effective communication, and these are prerequisites to community policies and a robust software release.

Deciding exactly how to deploy the SDKs at the facilities is itself a challenge. ECP applications will use different combinations of ST software in different configurations. For example, applications will want mathematical libraries capabilities from the xSDK build on top of both MPICH and OpenMPI, and will want different configurations of those mathematical libraries.

Solution Strategy The SDK solution strategy involves pursuing interoperability and sustainability goals by grouping ST software projects into logical collections whose members will benefit from a common set of community policies as well as increased communication between members to standardize approaches where sensible and establish better software practices. The SDK effort will also facilitate the use of common infrastructure, such as CI testing at the major computing facilities and the Spack [4] package manager.

SDK release and delivery goals will also benefit from common package manager and testing infrastructure. In addition to delivery through Spack, the SDK will also explore the option of binary delivery, possibly through OpenHPC.

Recognizing the different release readiness and broader maturity differences between different ECP ST products, the early release strategy is to include only those products ready for a joint release in the initial SDK release, but to also continue to work with other products in preparing for subsequent release opportunities.

Recent Progress The SDK team decided to use a horizontal as opposed to vertical approach for forming SDKs. A horizontal approach emphasizes the grouping of software according to function and purpose, such as MPI implementations, compiler frameworks, performance analysis, or mathematical libraries. A vertical approach would have grouped ST software products based on products that are built in a stack for end users.

Figure 65 illustrates the horizontal relationship between ST products in an SDK. The packages shown in green are all members of the xSDK, and are all mathematical libraries software. Software shown in black are not xSDK member packages. However, OpenMPI is an ECP ST software package. Although OpenMPI is commonly used with xSDK member packages (a vertical relationship), it is not a mathematical library (a horizontal relationship), and is therefore not part of the xSDK.

Some initial SDK groupings have been proposed based on the horizontal grouping approach. Lower level SDKs from the Programming Models and Runtimes area are available in a container environment for the purposes of supporting higher-level SDKs and testing SDK infrastructure.

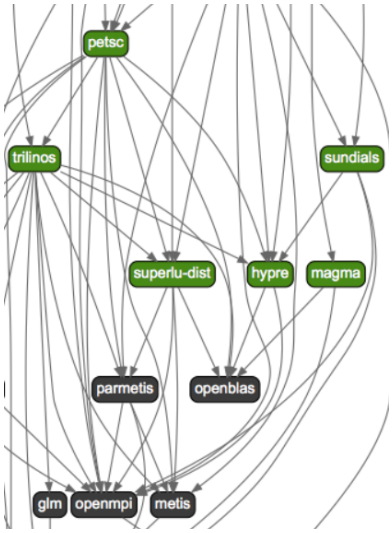


Figure 65: The above diagram shows a snippet of the Spack dependency tree including six xSDK member packages. While multiple xSDK member packages depend on OpenMPI, which is an ECP ST software product, OpenMPI is not part of the xSDK

Next Steps Current and near-term efforts include:

- defining additional ST SDKs based on horizontal grouping principles.
- beginning regular requirements discussions with computing facilities.
- identifying resources and a plan for assisting ECP ST packages with Spack package creation (over 50% already have a Spack package).
- beginning community policy discussions within SDK package teams.
- choosing a target set of ST products for inclusion in the FY19 Q1 release.

4.5.2 LANL ATDM Software Ecosystem & Delivery Projects - Resilience Subproject

Overview The resilience subproject has a goal of providing tools and solutions for users and administrators of the Exascale machines to deal with expected unreliable operations. Due to the extreme scale and bleeding-edge components necessary to meet the needs of the ECP project, it is expected that the applications will need to deal with application / job interrupts at a rate higher than users are used to on conventional extreme-scale systems of today. This also means that datacenter operations will be affected and modeling tools needs to be provided to monitor and be prepared for this situation.

In general, this subproject approaches this through a combination of data collection of today's extreme-scale systems (monitoring), data analytics, machine learning, model building, and fault injection into applications at scale.

Key Challenges The key challenges in this area are that it is difficult to acquire data on contemporary systems for a number of reasons such as NDA, security, and jitter / noise. We expect future systems will be complex enough that it is hard to estimate the parameters well and it must be understood that the modeling can only be as good as the input to the models. Care must be taken to get good input data from the vendors and the ECP project should place emphasis on getting quality monitoring data from around DOE datacenters that can seed modeling efforts on contemporary systems.

Reliability is a topic area which is sensitive not only amongst the government but also the vendors providing solutions and, as such, thought should be put into how we are to assemble information and disseminate for maximal progress of the project while also protecting the participants.

Solution Strategy This subproject's solution leverages data collection and analytics to do what is termed *data-driven datacenter design*. That is, using data from supercomputer field data (real and modeled from prospective / hypothetical machines) to design better datacenters and supercomputers themselves to better fit our users' needs. Further, we provide software evaluation tools to emulate soft faults so users can inject faults into their code and empirically evaluate their applications' resiliency and vulnerability to corruption. In this way, they can develop mitigation strategies and evaluate the efficacy of those approaches. It is important to understand, however, that these approaches are predicated on good input data that these faults are reasonable given what is expected for target Exascale computer platforms. The tools provided by this subproject can create nearly any fault model, but those faults need to be configured and driven by some information and while we can provide some initial suggestions it is theoretically possible that those would be incorrect.

Recent Progress The subproject recently completed a milestone where two supercomputer resiliency models were created. One is considerably more complex than the other but includes parameters which we found to be hard to measure on contemporary systems. It provides an example of features which we find desirable measurement points on future systems. The simplified model we provided we used on the Trinity supercomputer at LANL, the largest system at LANL. Due to the nature of the work performed on the system and the NDA-nature of the results of the analysis, we were unable to share publicly the results but did share the results in meetings with several laboratories and other government agencies, demonstrating the model and showed that it closely matches certain parameters from RFP (NDA) documents and neutron beam studies that have been published by the team.

Next Steps The FSEFI[197, 198] fault injector continues development toward a late year milestone of parallel fault injection demonstration. Additionally, there is an upcoming milestone for machine learning using data from supercomputer field data.

4.5.3 LANL ATDM Software Ecosystem & Delivery Projects - BEE/Charliecloud Subproject

Overview The BEE/Charliecloud subproject is creating software tools to increase portability and reproducibility of scientific applications on high performance and cloud computing platforms. Charliecloud [199] is an unprivileged Linux container runtime. It allows developers to use the industry-standard Docker [200] toolchain to containerize scientific applications and then execute them on unmodified DOE facility computing resources without paying any performance penalty. BEE [201] (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 Other HPC-focused container runtimes exist, such as NERSC's Shifter [202] and Singularity [203]. 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 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 The BEE/Charliecloud project is focusing first on providing support for containerized production LANL scientific applications across all of the existing LANL production HPC systems. The BEE/Charliecloud components required for production use at LANL will be documented, released and fully supported. Follow-on development will focus on expanding support to additional DOE platforms. This will mean supporting multiple hardware architectures, operating systems, resource managers, and storage subsystems. Support for alternative container runtimes, such as Docker, Shifter, and Singularity is planned.

Recent Progress Recent Charliecloud progress has been focusing on understanding and documenting best practices for running large scale MPI jobs using containerized runtimes. Additional work has been done to enhance support for using containers with GPUs. Charliecloud is available at https://github.com/hpc/charliecloud and was recently approved for inclusion in the next release of OpenHPC.

BEE currently has beta-level support for launching Charliecloud containers on LANL HPC systems. Automated BEE scalability testing is nearing production readiness.

Next Steps There is an Integrated Project Team (IPT) working at LANL on demonstrating a complete BEE/Charliecloud workflow using production applications on production HPC systems. This work will produce a well-documented and packaged BEE release consisting of the BEE-Charliecloud launcher, BEEflow workflow management system, and support for the Slurm resource manager.

4.5.4 2.3.5.03 LLNL ATDM SW Ecosystem & Delivery: DevRAMP

Overview The **DevRAMP** (Reproducibility, Analysis, Monitoring, and Productivity for Developers) is creating tools and services that multiply the productivity of HPC developers through automation. The project has two major software efforts:

1. **Spack** is a package manager for HPC [204, 205, 206, 4, 207, 208, 209, 210, 211, 212, 213, 214, 215]. It automates the process of downloading, building, and installing different versions of HPC software packages and their dependencies. With Spack, facilities can manage complex, multi-user software deployments on supercomputers, and developers can keep their own dependencies up to date in a single-user environment. Spack enables complex applications to be assembled from components, lowers barriers to software reuse, and allows complex HPC environments to be reproduced easily. It is the glue that holds the ECP software ecosystem together.

2. **Sonar** is a data cluster and a software stack for performance monitoring and analysis. Sonar allows developers and facility staff to understand how supercomputers and applications interact. The system continuously aggregates data from system-level monitoring services, application-level measurement tools, and facility power and temperature meters. Any facility user can log in, examine performance data and other data collected about their HPC jobs, and share it with other users. Our focus is on security for the services running on Sonar (e.g., JupyterHub), and on developing an easy-to-use query tool, which we call ScrubJay.

Key Challenges *Spack* makes HPC application complexity manageable. HPC simulation codes are notoriously complicated, and installing a code on a new platform is not a simple task. Codes are not self-contained; they rely on tens or even hundreds of *dependencies* to implement their functionality. While dependencies allow us to leverage the work of others through code reuse, integrating hundreds of dependencies in the same application is a monumental challenge. Dependencies may require different configurations and versions of their own dependency libraries, and codes must be built and tested with different implementations of interfaces MPI, BLAS, and LAPACK. The space of possible builds is combinatorial in size, and it can only be effectively managed through automation.

Sonar aims to find the root causes of performance variability of jobs that run at Livermore Computing. A single application execution may have a different runtime depending on the system it runs on, the file system it uses, or other jobs running at the same time. A single application execution may not accurately characterize performance. The causes of this variability may include file system performance, per-node power consumption, application code performance, and even processor performance from node to node. To analyze and understand performance variability, we must collect data from the HPC center *and* from jobs running at the facility, but we cannot show all application- and facility-level data to all users. We need hardened security and fine-grained permissions to ensure that data on Sonar is kept safe.

Solution Strategy *Spack* addresses HPC software complexity in three ways. First, it provides a powerful build system and a domain-specific language for writing package recipes. This allows HPC builds to be templated and easily mapped to HPC architectures. It provides dependency resolution infrastructure, including virtual dependencies, a full parameter system, and a dependency resolver we call a *concretizer*. Together, these allow developers to specify customize builds for their own environments based on *abstract* descriptions, while Spack handles the tedious aspects of integrating with a new machine. Finally, Spack is an online service where package recipes can be shared and the HPC community can leverage each others' effort to reuse software. One of Spack's greatest strengths is its very active online contributor community.

Sonar is a data cluster, a data model, a set of monitoring tools, and an analysis front-end for manipulating this data. To deploy this in the Livermore Computing (LC) center, our project has focused on securing all components of this stack so that they can be hosted as services in a multi-tenant environment. We have deployed the Cassandra database, the Spark parallel processing engine, a custom tool of our own called ScrubJay, among other components. We have added integration that allows each tool to use LC's user ids and groups with access control lists. In the case of Jupyter Notebooks and other services that launch jobs, we are also hardening the process launch infrastructure so that notebooks launch on behalf of users run *as* those users in the LC environment.

Recent Progress Figure 66 (left) shows the progress so far on Sonar deployment. The Cassandra database, as well as data collection from compute clusters, filesystems, and facility temperature data is complete. We have also recently completed a full-center deployment of JupyterHub for accessing the data in the cluster. We were able to deploy JupyterHub by adding SSL encryption to links between the hub and each notebook; this required us to build a custom certificate authority into JupyterHub. We are currently preparing to upstreaming our solution to the mainline JupyterHub repository.

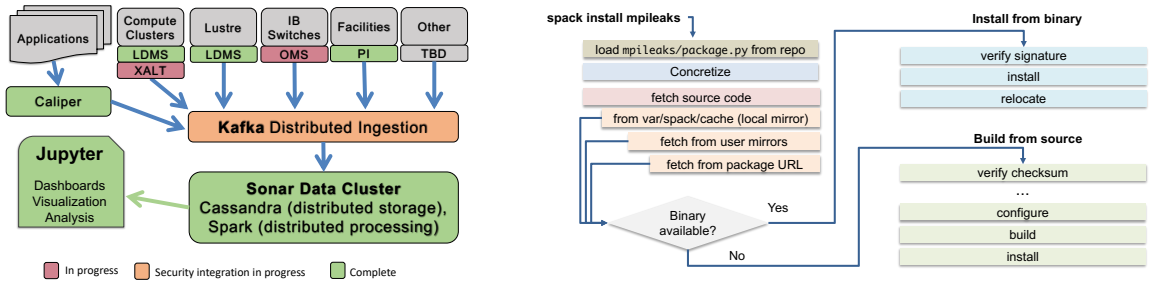


Figure 66: Sonar deployment (left) and Spack binary packaging (right).

We recently added the capability to create relocatable binary packages to Spack (Figure 66, right). Spack typically builds all of its packages from source, as this is the de-facto software distribution method for HPC systems. Recent additions to Spack allow us to package even HPC software in optimized binary form, and to install in minutes what could previously take hours. We are currently working on making Spack's dependency resolution algorithm more robust to handle more aggressive use of binary packages. Figure 67 shows the current Spack concretizer, which will only reuse binaries that exactly match a client-side concrete spec. The new version will allow Spack to consider available remote binaries at install time, and to search more aggressively through available binaries for usable dependencies. This will mean even less rebuilding of software, and for HPC systems, this means we will be able to match optimized binaries to the machine.

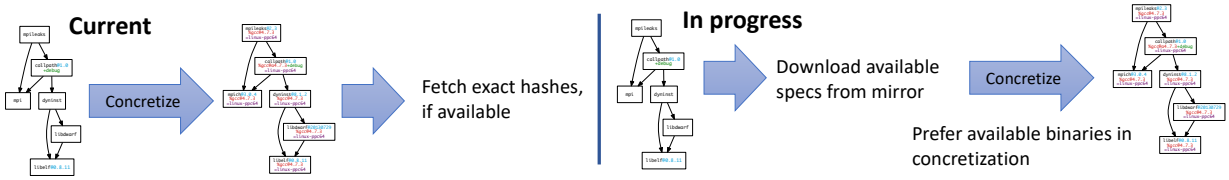


Figure 67: Spack concretizer design changes.

Next Steps

1. **Securing Kafka for Sonar.** Sonar will ingest application data from the filesystem using a tool called Kafka. Kafka is a streaming communication infrastructure; it runs as a daemon and reliably transfers data from files to a database. Kafka typically runs as a single user, but HPC centers are multi-tenant installations. We will be adding functionality to ensure that Kafka preserves the identity of file owners in the filesystem when it ingests data into the database.

2. **Spack release testing and binary hosting.** Spack serves as a repository for package recipes, but we do not have any automated build testing of packages included in the Spack repository. To increase the robustness and reliability of Spack, we are adding automation for building all packages in each Spack release, and for creating binary packages from such builds. We aim to provide binary packages for common configurations of Spack packages.

4.5.5 Sandia ATDM Software Ecosystem and Delivery - Technology Demonstrator

Overview This project is part of the NNSA/ASC program and is primarily focused on the integration of programming models technologies in existing applications. This project spans both the Code Development and Applications (CDA) and Architecture and Software Development (ASD) components of Sandia's ATDM program element, and this description only covers the ASD portion of the project.

The purpose of this project is to broaden the impact of programming model technologies developed in ATDM by integrating these technologies into an existing ASC code suite, evaluating the ability of these technologies to be incrementally added into a legacy code base and to deliver performance improvements on current- and next-generation computing platforms. In particular, this subproject focuses on using and evaluating Kokkos and asynchronous many task (AMT) programming models in the NGP Contact code. Kokkos is an abstraction layer that implements a parallel programming model in C++ for writing performance portable applications targeting all major HPC platforms. NGP Contact is a performance-critical proximity search component relevant to many NNSA applications, including ASC production applications that invoke mechanical or thermal contact, coupled physics transfers, and particle algorithms. This project is important the overall ECP efforts because it evaluates how well technologies developed for next-generation applications can be leveraged by existing large application code bases, such as the ASC Sierra toolkit, in an effort to maximize the impact of ECP technologies.

Key Challenges The main challenge associated with this project is the incremental integration of a new programming model and programming system into an existing very large production application. This challenge is similar in many ways to the incremental integration of OpenMP into an existing large MPI application. As Kokkos is a higher-level abstraction layer that can be implemented using OpenMP, there are additional challenges around the ability of Kokkos' abstractions and interfaces to integrate into an existing application and minimizing the performance penalty of the abstraction layer for different underlying hardware implementations.

Solution Strategy The strategy for this project is to produce demonstration applications that drive the development of an AMT scheduling toolset and to enable leveraging of Sierra-developed technologies to support ATDM application milestones.

Recent Progress The Sierra Solid Mechanics team helped integrate a GPU enabled bounding box search into the Technology Demonstrator (TD) application. This is the first component needed for a GPU contact algorithm and is the result of an ongoing collaboration between ATDM and Sierra/SM. The TD application is linking in this search and exercising it on both a CPU and GPU.

Next Steps The next component to integrate is a GPU enabled closest point projection, also developed jointly by ATDM and Sierra/SM.

4.5.6 Sandia ATDM Software Ecosystem and Delivery - OS/On-Node Runtime

Overview This project is part of the NNSA/ASC program and is primarily focused on operating system and runtime system (OS/R) technology development and evaluation. This project is tightly aligned with the Qthreads ECP project.

This project focuses on the design, implementation, and evaluation of OS/R interfaces, mechanisms, and policies supporting the efficient execution of the ATDM application codes on next-generation ASC 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 Key challenges for this project include:

- **Improve the understanding of the use of containers and virtualization technology to support ATDM applications and workloads** Containers are gaining popularity as a way to package applications and virtualize the underlying OS to allow a set of executables built for one platform to be run unmodified on a different platform. There are several different approaches to building and deploying containers, each with differing sets of capabilities and features.
- **Characterizing applications use of MPI and sensivity to system noise** Understanding how applications use MPI and its associated network resources requires both application- and hardware-level information that must be coordinated on time scales of less than a microsecond. It is also extremely difficult to isolate the sources of system noise and characterize the non-local side effects of unplanned detours that interrupt application execution flow.

Solution Strategy The strategy for containers and virtualization is to evaluate the different technology options using ATDM applications and workflows and compare the results against a set of evaluation criteria.

In order to characterize applications use of MPI and sensitivity to system noise, this project has developed a simulation environment that can be used to track MPI and network resource usage. This project is also using lightweight operating systems, which are virtually devoid of system noise, help understand how applications, especially those employing an ATM programming model, are impacted by OS noise.

Recent Progress Potential use cases and technical options for container technologies in ATDM were evaluated and published in a conference paper [216], which was nominated for best paper.

A journal article with analyses of MPI queue behavior observed during executions of Sandia mini-apps, as well as LAMMPS and CTH, has been accepted for publication [217]. The techniques were also applied to SPARC, and an expanded tech report version including the SPARC results will be available soon.

Next Steps Sandia's Kitten lightweight kernel has been ported to the Cray XC architecture and has been demonstrated on the Volta testbed system. Going forward, we are working on plans to port Kitten and/or the RIKEN McKernel lightweight kernel to the Vanguard Petascale ARM platform. We will be visiting RIKEN in early March to discuss plans and collaboration opportunities.

Close to having the Qthreads backend to Kokkos patched up after last year's frontend-to-backend API changes. Completion is contingent on resolution of the Kokkos team's remaining patches to the threads back-end, with which Qthreads shares common hooks.

We are participating in an ECP working group on container technology and using the results of evaluation to guide future activities in this area.

4.5.7 Argo

Overview The Argo project [218] is building portable, open source system software that improves the performance and scalability and provides increased functionality to Exascale applications and runtime systems.

We focus on four areas of the OS/R stack where the need from the ECP applications and facilities is perceived to be the most urgent: 1) support for hierarchical memory; 2) dynamic and hierarchical power management to meet performance targets; 3) containers for managing resources within a node; and 4) internode interfaces for collectively managing resources across groups of nodes.

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 [219]. At the same time, HPC hardware complexity increases as well, from deeper memory hierarchies encompassing on-package DRAM and byte-addressable NVRAM, to heterogeneous compute resources and performance changing dynamically based on power/thermal constraints.

To meet the emerging needs of ECP workloads while providing optimal performance and resilience, the compute, memory, and interconnect resources must be managed in cooperation with applications and runtime systems; yet existing resource management solutions lack the necessary capabilities and vendors are reluctant to innovate in this space in the absence of clear directions from the community.

Solution Strategy Our approach is to augment and optimize for HPC the existing open source offerings provided by vendors. We are working with ECP applications and runtime systems to distill the needed new interfaces and to build, test, and evaluate the newly implemented functionality with ECP workloads. This needs to be done in cooperation with facilities, who can provide early hardware testbeds where the newly implemented functionality can be demonstrated to show benefits, tested at scale, and matured. Over the years we have cultivated an excellent relationship with the vendors providing HPC platforms because our approach has been to augment and improve, rather than develop our own OS/R from scratch. IBM, Cray, and Intel are eager to integrate the components we develop for ECP that can help applications.

Our work in each area focuses on the following:

1. **Hierarchical memory:** Incorporate NVRAM into the memory hierarchy using UMap: a user-space `mmap` replacement for out-of-core data, leveraging recent `userfaultfd` mechanism of the Linux kernel for page fault handling, featuring application-class specific prefetching and eviction algorithms. Expose deep DRAM hierarchy by treating high-bandwidth memory (MCDRAM, HBM) as a scratchpad [220], managed by the Argonne Memory Library (AML), which provides applications with asynchronous memory migration between memory tiers and other convenience mechanisms.

2. **Power management:** *PowerStack* will explore hierarchical interfaces for power management at three specific levels [221, 222, 223, 224]: the global level of batch job schedulers (which we refer to as the Global Resource Manager or GRM), the enclave level of job-level runtime systems (open-source solution of Intel GEOPM and the ECP Power Steering project will be leveraged here), and the node-level through measurement and control mechanisms integrated with the NRM (described below). At the node level, we will develop low-level, vendor-specific monitoring/controlling capabilities to monitor power/energy consumption, core temperature and other hardware status [225, 226], and control the hardware power capping and the CPU frequencies.

3. **Containers:** Develop a Node Resource Manager (NRM) that leverages technologies underlying modern container runtimes (primarily `cgroups`) to partition resources on compute nodes [227], arbitrating between application components and runtime services.

4. **Hierarchical resource management:** Develop a set of distributed services and user-facing interfaces [228] to allow applications and runtimes to resize, subdivide, and reconfigure their resources inside a job. Provide the enclave abstraction: recursive groups of nodes that are managed as a single entity; those enclaves can then be used to launch new services or to create subjobs that can communicate with each other.

Recent Progress We identified initial representative ECP applications and benchmarks of interest, focusing in particular on characteristics such as: coupled codes consisting of multiple components, memory-intensive codes that do not fit well in DRAM or that are bandwidth-bound, and codes with dynamically changing resource requirements.

We designed and developed an API between Node Power and Node Resource Manager (NRM), which in turn allows Global Resource Manager (GRM) to control and monitor power and other node-local resources. Additionally, we studied the effect of power capping on different applications using the NodePower API and developed power regression models required for a demand-response policy, which may be added to the GRM in future.

We developed Yggdrasil, a resource event reactor that provides the enclave abstraction. It can use standard HPC schedulers such as SLURM or Flux as a backend.

We developed the initial, centralized version of the Global Resource Manager (GRM) providing a hierarchical framework for power control. We enhanced existing PowSched codebase to use the power controller in the node OS to acquire node local power data and adjust the power caps. We also developed a variation-aware scheduler to address manufacturing variability under power constraints with Flux infrastructure, and extended SLURM to support power scheduling plugins to enable our upcoming GRM milestones.

We designed and developed the first version of the unified Node Resource Manager. The NRM provides high level of control over node resources, including initial allocation at job launch and dynamic reallocation at the request of the application and other services. The initial set of managed resources includes CPU cores and memory; they can be allocated to application components via a container abstraction, which is used to describe partitions of physical resources (to decrease interference), and more.

We developed the first stable version of UMap, the user-space memory map page fault handler for NVRAM. UMap handler maps application threads' virtual address ranges to persistent data sets, transparently pages in active pages and evicts unused pages. We evaluated the costs and overheads of various approaches and characterized end-to-end performance for simple I/O intensive applications.

We designed AML, a memory library for explicit management of deep memory architectures, and validated it on Intel's Knights Landing. Its main feature is a scratchpad API, allowing applications to implement algorithms similar to out-of-core for deep memory. We provided multiple optimized versions of memory migration facilities, ranging from a regular copy to a transparent move of memory pages, using synchronous and asynchronous interfaces and single- and multithreaded backends.

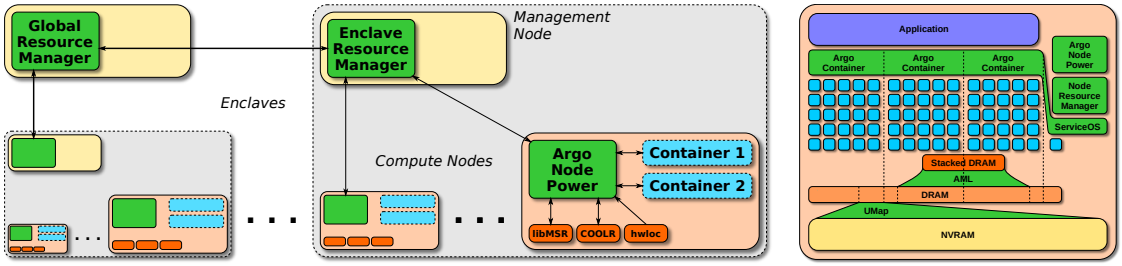


Figure 68: Global and node-local components of the Argo software stack and interactions between them and the surrounding HPC system components.

Next Steps We plan to investigate different power management policies, particularly demand response, which is becoming a crucial feature for data centers, allowing to reduce the power consumption of servers without killing running applications or shutting down nodes. We plan to continue the development of power-aware versions of SLURM and Flux, and enable the path toward integration with GEOPM and NRM.

We plan to improve the control loop inside the NRM to take into account I/O and memory interference when placing containers on a node. We are also working on making the NRM implementation compatible with Singularity.

We plan to improve the performance of UMap and demonstrate it with an astronomy application having hundreds of files. We plan to port applications to AML and integrate with runtime projects like BOLT or PaRSEC for efficient use of Knights Landing memory using our scratchpad.

4.5.8 Flang

Overview The Flang project provides an open source Fortran [229] [230] [231] compiler licensed and designed for integration with the LLVM Compiler Infrastructure (see http://llvm.org) [232]. The Flang compiler is a cross-platform Fortran solution for multicore CPUs available to ECP and HPC users today. Goals of the project include extending support to GPU accelerators and Exascale systems, and supporting LLVM-based software and tools R&D of interest to a large deployed base of Fortran applications. With the growing popularity and wide adoption of LLVM within the broader HPC community, this project provides the foundation for a Fortran solution that will complement and interoperate with the Clang/LLVM C++ compiler. It will allow Fortran to grow into a modernized open source form that is stable and has an active footprint within the LLVM community, and will meet the needs of a broad scientific computing community. The Flang source code base is derived from the well-established PGI/NVIDIA proprietary Fortran compiler, which provides a solid compiler base from which to evolve a Fortran front-end and runtime libraries coded in a style which will allow new contributors to ramp up and be productive quickly. The most active developers continue to be PGI/NVIDIA, with recent contributions from ARM Ltd and the DOE National Labs demonstrating a path to long-term sustainability. Full source is available at https://github.com/flang-compiler/flang, and mailing lists and a Slack channel have been created (http://lists.flang-compiler.org/ and http://flang-compiler.slack.com/ respectively).

Key Challenges There are several commercially-supported Fortran compilers, typically available on only one or a few platforms. None of these are open source. The GNU gfortran open source compiler is available on a wide variety of platforms, but the source base is not modern LLVM-style C++, and the GPL open source license is not compatible with LLVM. As a result, leveraging gfortran in the LLVM community is problematic. The primary challenge of this project is to create a source base with the maturity, features and performance of proprietary solutions, the cross-platform capability of GNU compilers, and which is licensed and coded in a style that will be embraced by the LLVM community. Another key challenge is robustly supporting all Fortran language features, programming models and scalability that will be required for effective use on Exascale systems, and balancing project resources across immediate bug fixes and minor requests-for-enhancement versus the strategic requirement to modernize the source code base.

Solution Strategy The Flang project strategy was and is to streamline (remove unused legacy code, modify source to compile cleanly with no Clang warnings, minimize or eliminate use of conditional compilation, incremental migration to C++) and integrate the existing PGI Fortran front-end with LLVM's opt/llc components to create an open source baseline compiler with the same Fortran language and OpenMP features as the commercial PGI Fortran compiler. From that baseline compiler, additional Fortran language and OpenMP features will be added while tracking the evolution of LLVM closely from release-to-release. Also from that baseline compiler, selected components will be prioritized for significant refactoring or re-writing as required to meet the requirements of the LLVM community, to enable LLVM developers to easily ramp up and contribute, and to support all of modern Fortran and its related programming models in a fully robust way across all supported platforms.

Recent Progress Recent developments have added support for Fortran debugging, substantially all of OpenMP 4.5 [233] targeting multicore CPUs, and an all-new infrastructure for optimized scalar and SIMD Fortran numerical intrinsics. With these in place, Flang delivers performance averaging 5% faster than gfortran across a range of benchmarks and enables ECP developers to write and test OpenMP 4.5 on multicore CPU targets.

Figure 69 illustrates the relative performance of Flang against gfortran.

Next Steps We have recently started work on OpenMP 4.5 target offload support for NVIDIA Tesla GPUs and on an all-new Fortran 2018 parser written in modern C++ suitable for the LLVM project. We will continue work on GPU targeting for the remainder of 2018, and after DOE review we will open source the new Fortran 2018 parser and begin work on related components including semantic analysis, OpenMP support in the new parser, and the aspects of the front-end required for lowering representations and bridging to

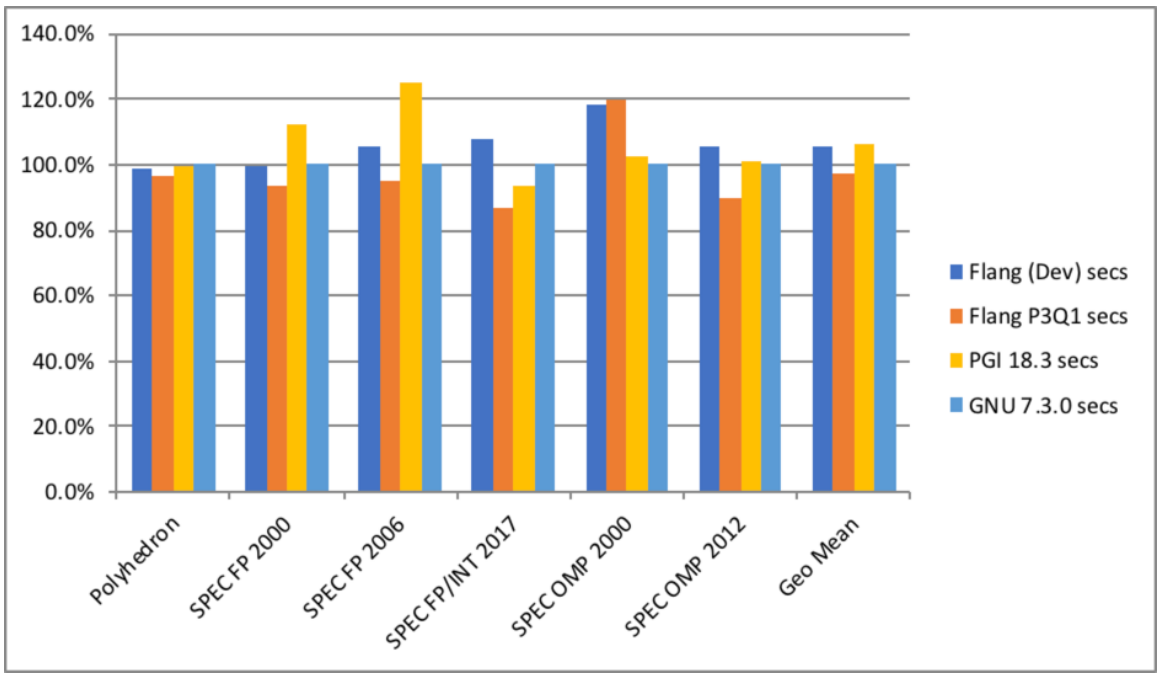


Figure 69: Flang performance is benchmarked against other Fortran compilers. The above diagram shows the relative performance of Flang against PGI Fortran and GNU gfortran

LLVM. As we work through these aspects of modernization, we will continue to build up the Flang testing infrastructure and be as responsive as possible to bug reports and minor requests for enhancement from what will hopefully be a growing base of Flang users in the ECP and HPC communities.

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.

ACKNOWLEDGEMENTS

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] Rajeev Thakur, Pat McCormick, Jim Ahrens, Al Geist, Michael A. Heroux, Rob Neely, Rob Ross, Martin Schulz, and Jeff Vetter. Ecp software technology gap analysis. This is an internal report for the Exascale Computing Project, Software Technology Focus Area.

[2] 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.

[3] Roscoe Bartlett, Irina Demeshko, Todd Gamblin, Glenn Hammond, Michael Heroux, Jeffrey Johnson, Alicia Klinvex, Xiaoye Li, Lois McInnes, J. David Moulton, Daniel Osei-Kuffuor, Jason Sarich, Barry Smith, James Willenbring, and Ulrike Meier Yang. xsdk foundations: Toward an extreme-scale scientific software development kit. *Supercomput. Front. Innov.: Int. J.*, 4(1):69–82, March 2017.

[4] 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.

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

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

[7] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[8] *hypre*: High performance preconditioners. http://www.llnl.gov/CASC/hypre/.

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

[10] Marat Valiev, Eric J Bylaska, Niranjan Govind, Karol Kowalski, Tjerk P Straatsma, Hubertus JJ Van Dam, Dunyou Wang, Jarek Nieplocha, Edoardo Apra, Theresa L Windus, et al. Nwchem: a comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477–1489, 2010.

[11] Mark S Gordon and Michael W Schmidt. Advances in electronic structure theory: GAMESS a decade later. In *Theory and applications of computational chemistry*, pages 1167–1189. Elsevier, 2005.

[12] Bruce Palmer, William Perkins, Yousu Chen, Shuangshuang Jin, David Callahan, Kevin Glass, Ruisheng Diao, Mark Rice, Stephen Elbert, Mallikarjuna Vallem, et al. GridPACK: a framework for developing power grid simulations on high performance computing platforms. In *Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, pages 68–77. IEEE Press, 2014.

[13] Jeongnim Kim, Andrew Baczewski, Todd Beaudet, Anouar Benali, Chandler Bennett, Mark Berrill, Nick Blunt, Edgar Josue Landinez Borda, Michele Casula, David Ceperley, Simone Chiesa, bryan K clark, Raymond Clay, Kris Delaney, Mark Dewing, Ken Esler, Hongxia Hao, Olle Heinonen, Paul R C Kent, Jaron T. Krogel, Ilkka Kylanpaa, Ying Wai Li, M. Graham Lopez, Ye Luo, Fionn Malone, Richard Martin, Amrita Mathuriya, Jeremy McMinis, Cody Melton, Lubos Mitas, Miguel A. Morales, Eric Neuscamman, William Parker, Sergio Flores, Nichols A Romero, Brenda Rubenstein, Jacqueline Shea, Hyeondeok Shin, Luke Shulenburger, Andreas Tillack, Joshua Townsend, Norman Tubman, Brett van der Goetz, Jordan Vincent, D. ChangMo Yang, Yubo Yang, Shuai Zhang, and Luning Zhao. QMCPACK : An open source ab initio quantum monte carlo package for the electronic structure of atoms, molecules, and solids. *Journal of Physics: Condensed Matter*, 2018.

[14] Jeff Daily, Abhinav Vishnu, Bruce Palmer, Hubertus van Dam, and Darren Kerbyson. On the suitability of mpi as a PGAS runtime. In *High Performance Computing (HiPC), 2014 21st International Conference on*, pages 1–10. IEEE, 2014.

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

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

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

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

[19] 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.

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

[21] 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.

[22] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, T. Hoefler, S. Kumar, E. Lusk, R. Thakur, and J. L. Traeff. MPI on Millions of Cores. *Parallel Processing Letters (PPL)*, 21(1):45–60, March 2011.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] S. Seo, A. Amer, P. Balaji, C. Bordage, G. Bosilca, A. Brooks, P. Carns, A. Castello, D. Genet, T. Herault, S. Iwasaki, P. Jindal, L. V. Kale, S. Krishnamoorthy, J. Lifflander, H. Lu, E. Meneses, M. Snir, Y. Sun, K. Taura, and P. Beckman. Argobots: A lightweight low-level threading and tasking framework. *IEEE Transactions on Parallel and Distributed Systems*, 29(3):512–526, March 2018.

[40] Sean Treichler, Michael A. Bauer, Ankit V. Bhagatwala, Giulio Borghesi, Ramanan Sankaran, Hemanth Kolla, Jeffrey M. Larkin, Elliott Slaughter, Wonchan Lee, Alex Aiken, Jacqueline H. Chen, and Patrick S. McCormick. S3D-Legion: An Exascale Software for Direct Numerical Simulation of Turbulent Combustion with Complex Multicomponent Chemistry. In T.P. Straatsma, K.B. Antypas, and T.J. Williams, editors, *Exascale Scientific Applications: Scalability and Performance Portability*, Chapman & Hall/CRC Computational Science. CRC Press, 11 2017.

[41] 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. talk at ExaMPI 2017 workshop, Denver, CO, November 2017.

[42] D. Eberius, T. Patinyasakdikul, and G. Bosilca. Using Software-Based Performance Counters to Expose Low-Level Open MPI Performance Information. In *Proceedings of the 24th European MPI Users' Group Meeting*, page Article No. 7, Chicago, IL, September 2017.

[43] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, et al. The international exascale software project roadmap. *International Journal of High Performance Computing Applications*, 25(1):3–60, 2011.

[44] Bronis R. De Supinski and Michael Klemm. OpenMP Technical Report 6: Version 5.0 Preview 2. http://www.openmp.org/wp-content/uploads/openmp-tr6.pdf, 2017. [Online; accessed 13-April-2018].

[45] 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.

[46] Swaroop S Pophale, David E Bernholdt, and Oscar R Hernandez. Openmp 4.5 validation and verification suite, version 00, 12 2017.

[47] 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.

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

[49] 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.

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

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

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

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

[54] 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.

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

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

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

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

[59] 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.

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

[61] 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.

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

[63] 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.

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

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

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

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

[68] R. S. Engelschall. Portable multithreading - the signal stack trick for user-space thread creation. In *ATC*, 2000.

[69] Boost.Context. http://www.boost.org/doc/libs/1_57_0/libs/context/, 2009.

[70] 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.

[71] 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.

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

[73] 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://ieeexplore.ieee.org/document/6877339/.

[74] Dan Bonachea and Paul Hargrove. GASNet specification, v1.8.1. Technical Report LBNL-2001064, Lawrence Berkeley National Laboratory, August 2017. http://escholarship.org/uc/item/03b5g0q4.

[75] 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.

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

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

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

[79] GASNet website. http://gasnet.lbl.gov/.

[80] Paul H. Hargrove and Dan Bonachea. GASNet-EX performance improvements due to specialization for the Cray Aries network. Technical Report LBNL-2001134, Lawrence Berkeley National Laboratory, March 2018. http://escholarship.org/uc/item/6bn019rq.

[81] K.B. Wheeler, R.C. Murphy, and D. Thain. Qthreads: An API for programming with millions of lightweight threads. In *IEEE International Symposium on Parallel and Distributed Processing Workshops, 2008. IPDPSW 2008*, pages 1–8, April 2008.

[82] Sean Williams, Latchesar Ionkov, and Michael Lang. NUMA distance for heterogeneous memory. In *Proceedings of the Workshop on Memory Centric Programming for HPC*, MCHPC'17, pages 30–34, New York, NY, USA, 2017. ACM.

[83] Thaleia Dimitra Doudali and Ada Gavrilovska. CoMerge: Toward efficient data placement in shared heterogeneous memory systems. In *Proceedings of the International Symposium on Memory Systems*, MEMSYS '17, pages 251–261, New York, NY, USA, 2017. ACM.

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

[85] George Stelle, William S. Moses, Stephen L. Olivier, and Patrick McCormick. Openmpir: Implementing openmp tasks with tapir. In *Proceedings of the Fourth Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM-HPC'17, pages 3:1–3:12, New York, NY, USA, 2017. ACM.

[86] 2018 European LLVM Developers Meeting. 2018 European LLVM Developers Meeting. https://llvm.org/devmtg/2018-04/.

[87] S. K. Gutiérrez, K. Davis, D. C. Arnold, R. S. Baker, R. W. Robey, P. McCormick, D. Holladay, J. A. Dahl, R. J. Zerr, F. Weik, and C. Junghans. Accommodating thread-level heterogeneity in coupled parallel applications. In *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 469–478, May 2017.

[88] Flang Fortran Compiler. Flang Fortran Compiler. https://github.com/flang-compiler/flang.

[89] Dong H. Ahn Joachim Protze, Martin Schulz and Matthias S. Muller. Thread-local concurrency: A technique to handle data race detection at programming model abstraction. In *ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC 2018)*, 2018. To appear.

[90] Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamarić, Ignacio Laguna, Greg L. Lee, and Dong H. Ahn. Sword: A bounded memory-overhead detector of OpenMP data races in production runs. In *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018. To appear.

[91] Geoffrey Sawaya, Michael Bentley, Ian Briggs, Ganesh Gopalakrishnan, and Dong H. Ahn. FLiT: Cross-platform floating-point result-consistency tester and workload. In *2017 IEEE International Symposium on Workload Characterization (IISWC)*, pages 229–238, 2017.

[92] Andreas Krall and Thomas R. Gross, editors. *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2018, Vienna, Austria, February 24-28, 2018*. ACM, 2018.

[93] A. Haidar, H. Jagode, A. YarKhan, P. Vaccaro, S. Tomov, and J. Dongarra. Power-aware computing: Measurement, control, and performance analysis for intel xeon phi. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, Sept 2017.

[94] A. Haidar, H. Jagode, P. Vaccaro, A. YarKhan, S. Tomov, and J. Dongarra. Investigating power capping toward energy-efficient scientific applications. *Concurrency and Computation: Practice and Experience 2018;e4485*, pages 1–14, March 2018.

[95] 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.

[96] 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.

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

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

[99] 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.

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

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

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

[103] B. Smith, R. Bartlett, and xSDK developers. xSDK community package policies, 2017. version 0.3.0, November 6, 2017, https://dx.doi.org/10.6084/m9.figshare.4495136.

[104] R. Bartlett, J. Sarich, B. Smith, T. Gamblin, and xSDK developers. xSDK community installation policies: GNU Autoconf and CMake options, 2017. version 0.3.0, November 6, 2017, https://dx.doi.org/10.6084/m9.figshare.4495133.

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

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

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

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

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

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

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

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

[113] Alicia Marie Klinvex. xSDKTrilinos user manual. Technical Report SAND2016-3396 O, Sandia, 2016.

[114] 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.

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

[116] 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.

[117] Ben Bergen, Irina Demeshko, and Nick Moss. FleCSI: The flexible computational science infrastructure. Technical report, Los Alamos National Laboratory, 2015.

[118] 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*, SC '12, pages 66:1–66:11, Los Alamitos, CA, USA, 2012. IEEE Computer Society Press.

[119] Laxmikant V. Kale and Sanjeev Krishnan. Charm++: A portable concurrent object oriented system based on c++. In *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*, OOPSLA '93, pages 91–108, New York, NY, USA, 1993. ACM.

[120] Laxmikant V. Kale and Sanjeev Krishnan. CHARM++: A portable concurrent object oriented system based on c++. *SIGPLAN Not.*, 28(10):91–108, October 1993.

[121] CORPORATE The MPI Forum. MPI: A message passing interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Supercomputing '93, pages 878–883, New York, NY, USA, 1993. ACM.

[122] Anders Logg and Garth N. Wells. DOLFIN: Automated finite element computing. *ACM Trans. Math. Softw.*, 37(2):20:1–20:28, April 2010.

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

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

[125] 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.

[126] 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.

[127] 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.

[128] 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.

[129] 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.

[130] 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.

[131] 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.

[132] 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.

[133] SUNDIALS Project Team. SUNDIALS Web page. http://computation.llnl.gov/projects/sundials.

[134] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, 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 3.9. Technical Memorandum ANL-95/11 – Revision 3.9, Argonne National Laboratory, 2018.

[135] A. Dener, T. Munson, J. Sarich, S. Wild, S. Benson, and L. Curfman McInnes. TAO 3.9 Users Manual. Technical Memorandum ANL/MCS-TM-322 – Revision 3.9, Argonne National Laboratory, 2018.

[136] *A robust and scalable preconditioner for indefinite systems using hierarchical matrices and randomized sampling*, Orlando, USA, May 29 - June 2 2017.

[137] Y. Liu, M. Jacquelin, P. Ghysels, and X.S. Li. Highly scalable distributed-memory sparse triangular solution algorithms. In *Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*, Bergen, Norway, June 6-8, 2018 2018.

[138] P. Sao, R. Vuduc, and X.S. Li. A communication-avoiding 3d factorization for sparse matrices. In *32nd IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Vancouver, Canada, May 21-25 2018.

[139] A. Azad, A. Buluc, X.S. Li, X. Wang, and J. Langguth. A distributed-memory approximation algorithm for maximum weight perfect bipartite matching, January 2018. available via https://arxiv.org/abs/1801.09809.

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

[141] 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.

[142] 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.

[143] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.

[144] Michael W Mahoney et al. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[145] Jakub Kurzak, Mark Gates, Asim YarKhan, Ichitaro Yamazaki, Panruo Wu, Piotr Luszczek, Jamie Finney, and Jack Dongarra. SLATE working note 5: Parallel BLAS performance report. Technical Report ICL-UT-18-01, Innovative Computing Laboratory, University of Tennessee, March 2018. revision 03-2018.

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

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

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

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

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

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

[152] Production-ready, Exascale-enabled Krylov Solvers for Exascale Computing (PEEKS) icl.utk.edu/peeks/.

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

[154] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Kitware Inc., fourth edition, 2004. ISBN 1-930934-19-X.

[155] 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.

[156] 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.

[157] Woody Austin, Grey Ballard, and Tamara G. Kolda. Parallel tensor compression for large-scale scientific data. In *IPDPS'16: Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium*, pages 912–922, May 2016.

[158] Micah Howard, Andrew Bradley, Steven W. Bova, James Overfelt, Ross Wagnild, Derek Dinzl, Mark Hoemmen, and Alicia Klinvex. Towards performance portability in a compressible cfd code. In *AIAA Computational Fluid Dynamics Conference*, AIAA AVIATION Forum, 2017.

[159] NaluCFD webpage. http://nalucfd.org/.

[160] M. A. Gallis, J. R. Torczynski, S. J. Plimpton, D. J. Rader, and T. Koehler. Direct simulation monte carlo: The quest for speed. In *29th Intl Symposium on Rarefied Gas Dynamics*, AIP Conference Proceedings, 2014. http://sparta.sandia.gov.

[161] TuckerMPI. https://gitlab.com/tensors/TuckerMPI.

[162] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey. Fast Sort on CPUs and GPUs: A Case for Bandwidth Oblivious SIMD Sort. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10)*, pages 351–362, 2010.

[163] Leonardo Bautista-Gomez, Seiji Tsuboi, Dimitri Komatitsch, Franck Cappello, Naoya Maruyama, and Satoshi Matsuoka. FTI: High performance fault tolerance interface for hybrid systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11. ACM, 2011.

[164] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis de Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proceedings of the 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, 2010.

[165] 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.

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

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

[168] Peter Lindstrom and Markus Salasoo. ZFP version 0.5.3, 2018. https://github.com/LLNL/zfp.

[169] Peter Lindstrom. ZFP version 0.5.3 documentation, 2018. http://zfp.readthedocs.io/en/release0.5.3/.

[170] 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.

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

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

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

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

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

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

[177] 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.

[178] 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.

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

[180] 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.

[181] 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.

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

[183] 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.

[184] 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.

[185] 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.

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

[187] 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.

[188] 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.

[189] 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.

[190] 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.

[191] 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.

[192] Robert Miller, Kenneth Moreland, and Kwan-Liu Ma. Finely-threaded history-based topology computation. In *Eurographics Symposium on Parallel Graphics and Visualization*, pages 41–48, 2014. DOI 10.2312/pgv.20141083.

[193] James Reinders. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly, July 2007. ISBN 978-0-596-51480-8.

[194] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP*. MIT Press, 2007. ISBN 978-0-262-53302-7.

[195] B. Smith, R. Bartlett, and xSDK developers. xSDK community package policies, 2016. version 0.3, December 2, 2016, https://dx.doi.org/10.6084/m9.figshare.4495136.

[196] R. Bartlett, J. Sarich, B. Smith, T. Gamblin, and xSDK developers. xSDK community installation policies: GNU Autoconf and CMake options, 2016. version 0.1, December 19, 2016, https://dx.doi.org/10.6084/m9.figshare.4495133.

[197] PFSEFI. Parallel Fine-grained Soft Error Fault Injector (P-FSEFI). https://github.com/lanl/PFSEFI".

[198] Q. Guan, N. Debardeleben, S. Blanchard, and S. Fu. F-sefi: A fine-grained soft error fault injection tool for profiling application vulnerability. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 1245–1254, May 2014.

[199] 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.

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

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

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

[203] 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.

[204] Mario Melara, Todd Gamblin, Gregory Becker, Robert French, Matt Belhorn, Kelly Thompson, Peter Scheibel, and Rebecca Hartman-Baker. Using Spack to Manage Software on Cray Supercomputers. In *Cray Users Group (CUG 2017)*, Seattle, WA, May 7-11 2017.

[205] 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.

[206] 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.

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

[208] 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.

[209] Todd Gamblin. Decluttering HPC Software Chaos with SPACK. In *SIAM Conference on Parallel Processing for Scientific Computing (PP'18)*, Minisymposium on Productive Programming using Parallel Models, Tools and Scientific Workflows, Tokyo, Japan, March 7 2018.

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

[211] Todd Gamblin. How compilers affect dependency resolution in Spack. In *Free and Open source Software Developers' European Meeting (FOSDEM'18)*, Brussels, Belgium, February 3 2018. LLNL-PRES-745770.

[212] Todd Gamblin. Binary packaging for HPC with Spack. In *Free and Open source Software Developers' European Meeting (FOSDEM'18)*, Brussels, Belgium, February 4 2018. LLNL-PRES-745747.

[213] Todd Gamblin. Spack State of the Union. In *Exascale Computing Project 2nd Annual Meeting*, Knoxville, TN, February 6-8, 2018 2018. LLNL-PRES-746210.

[214] Todd Gamblin. Recent developments in Spack. In *Easybuild User Meeting*, Amsterdam, The Netherlands, January 30 2018.

[215] Gregory Becker, Peter Scheibel, Matthew P. LeGendre, and Todd Gamblin. Managing Combinatorial Software Installations with Spack. In *Second International Workshop on HPC User Support Tools (HUST'16)*, Salt Lake City, UT, November 13 2016.

[216] Andrew Younge, Kevin Pedretti, Ryan E. Grant, and Ron Brightwell. A tale of two systems: Using containers to deploy HPC applications on supercomputers and clouds. In *Proceedings of the 9th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, December 2017.

[217] Kurt Ferreira, Scott Levy, Kevin Pedretti, and Ryan E. Grant. Characterizing MPI matching via trace-based simulation. *Journal of Parallel Computing*, 2018. To appear.

[218] 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.

[219] Matthieu Dreher, Swann Perarnau, Tom Peterka, Kamil Iskra, and Pete Beckman. In situ workflows at exascale: System software to the rescue. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, pages 22–26. ACM, 2017.

[220] Swann Perarnau, Judicael A Zounmevo, Balazs Gerofi, Kamil Iskra, and Pete Beckman. Exploring data migration for future deep-memory many-core systems. In *IEEE Cluster*, 2016.

[221] D. Ellsworth, T. Patki, S. Perarnau, S. Seo, A. Amer, J. Zounmevo, R. Gupta, K. Yoshii, H. Hoffman, A. Malony, M. Schulz, and P. Beckman. Systemwide power management with Argo. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1118–1121, 2016.

[222] Daniel Ellsworth, Tapasya Patki, Martin Schulz, Barry Rountree, and Allen Malony. A unified platform for exploring power management strategies. In *Proc. 4th International Workshop on Energy Efficient Supercomputing*, 2016.

[223] Tapasya Patki, Natalie Bates, Girish Ghatikar, Anders Clausen, Sonja Klingert, Ghaleb Abdulla, and Mehdi Sheikhalishahi. Supercomputing centers and electricity service providers: A geographically distributed perspective on demand management in Europe and the United States. In Julian M. Kunkel, Pavan Balaji, and Jack Dongarra, editors, *International Supercomputing Conference (High Performance Computing), ISC-HPC*, 2016.

[224] Ryuichi Sakamoto, Thang Cao, Masaaki Knoda, Koji Inoue, Masatsugu Ueda, Tapasya Patki, Daniel Ellsworth, Barry Rountree, and Martin Schulz. Production hardware overprovisioning: Real-world performance optimization using an extensible power-aware resource management framework. In *Proc. 31st Internation Parallel and Distributed Processing Symposium*, 2017.

[225] Kazutomo Yoshii, Pablo Llopis, Kaicheng Zhang, Yingyi Luo, Seda Ogrenci-Memik, Gokhan Memik, Rajesh Sankaran, and Pete Beckman. Addressing thermal and performance variability issues in dynamic processors, 3 2017.

[226] Kaicheng Zhang, Seda Ogrenci-Memik, Gokhan Memik, Kazutomo Yoshii, Rajesh Sankaran, and Pete Beckman. Minimizing thermal variation across system components. In *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, pages 1139–1148. IEEE, 2015.

[227] Judicael A Zounmevo, Swann Perarnau, Kamil Iskra, Kazutomo Yoshii, Roberto Gioiosa, Brian C Van Essen, Maya B Gokhale, and Edgar A Leon. A container-based approach to OS specialization for exascale computing. In *Containers, 1st Workshop on (WoC)*, 2015.

[228] Swann Perarnau, Rajeev Thakur, Kamil Iskra, Ken Raffenetti, Franck Cappello, Rinku Gupta, Pete Beckman, Marc Snir, Henry Hoffmann, Martin Schulz, et al. Distributed monitoring and management of exascale systems in the Argo project. In *Distributed Applications and Interoperable Systems*, pages 173–178. Springer International Publishing, 2015.

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

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

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

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

[233] OpenMP Architecture Review Board. OpenMP Application Program Interface Version 4.5. Specification, OpenMP.org, 2015.