LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-JRNL-872481

ECP Libraries and Tools: An Overview

M. A. Heroux, L. C. McInnes, J. Ahrens, T. Gamblin, T. C. Germann, X. S. Li, K. Mohror, T. Munson, S. Shende, R. Thakur, J. Vetter, J. Willenbring

February 14, 2025

The International Journal of High Performance Computing Applications

Disclaimer

ECP Libraries and Tools: An Overview

Michael A. Heroux[1], Lois Curfman McInnes[2], James Ahrens[3], Todd Gamblin[4], Timothy C. Germann[3], Xiaoye Sherry Li[5], Kathryn Mohror[4], Todd Munson[2], Sameer Shende[6], Rajeev Thakur[2], Jeffrey Vetter[7], and James Willenbring[1]

Abstract

The Exascale Computing Project (ECP) Software Technology and Co-Design teams addressed the growing complexities in high-performance computing (HPC) by developing scalable software libraries and tools that leverage exascale system capabilities. As we enter the exascale era, the need for reusable, optimized software solutions that can handle the unique challenges posed by these systems becomes increasingly important. The primary challenges the ECP teams faced were to create software libraries and tools that are performant on exascale architectures and portable and usable across diverse hardware platforms. Efforts addressed issues related to concurrent execution, memory management, and the integration of heterogeneous computing resources, such as GPUs from multiple vendors. The ECP's strategy involved a structured development process encompassing the creation, optimization, and deployment of software in collaboration with industry, academia, and national laboratories. The project was organized into several technical areas: co-design of domain-specific suites with target applications, programming models and runtimes, development tools, mathematical libraries, data and visualization tools, and software ecosystem and delivery mechanisms. ECP has successfully developed a large portfolio of software libraries and tools that demonstrate significant improvements in performance and scalability on exascale systems. These products have been integrated into the Department of Energy's computing facilities, supporting various scientific applications and ensuring robust performance across different hardware setups. ECP advancements in software development for exascale computing highlight the importance of a collaborative and adaptive approach to handling next-generation HPC systems complexities. The lessons learned emphasize the need for continuous engagement with end-users and vendors, and the importance of maintaining a balance between innovation and practical implementation. Future efforts will focus on ensuring scalability, keeping pace with rapid hardware advancements, and further enhancing the interoperability and usability of the software ecosystem. Subsequent articles in this special issue provide in-depth discussions and case studies into specific library and tool efforts.

Keywords

ECP, software, libraries, tools, software ecosystem

Contents

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

Corresponding author:
Michael A. Heroux, Sandia National Laboratories, PO Box 5800, Albuquerque, NM 87185, USA.
Email: mheroux@acm.org

Introduction

In 2016, the United States Department of Energy (DOE) initiated the U.S. Exascale Computing Project (ECP) (Kothe et al. 2019),* with the organizational structure shown in Figure 1. ECP efforts produced dozens of scalable applications (Alexander et al. 2020), supporting software libraries and tools (Heroux et al. 2022), and other technology investments that led to the successful demonstration of the Frontier exascale system, capable of performing just over one exaflop, or a billion-billion (10^{18}) calculations per second. In addition, the Aurora and El Capitan systems at Argonne and Lawrence Livermore National Laboratories are well on the way to completion, slated to surpass Frontier once completed. ECP, as a project, reached its completion in December 2023.

ECP's inception was driven by objectives critical to both scientific advancement and national security. The computational power of exascale systems was targeted to facilitate breakthroughs in various research domains, including climate science, materials science, energy sustainability, and national security missions. ECP was sponsored by the DOE's Office of Science and the National Nuclear Security Administration and executed across the DOE laboratory complex, with important contributions from academic partners and private industry.

Among ECP's significant achievements has been the development and optimization of specialized software and algorithms in the form of reusable libraries and tools designed to exploit the capabilities of exascale computing platforms. As indicated in Figure 1, these reusable libraries and tools were developed by ECP Software Technology and Co-Design teams. In this paper, we provide an overview of ECP libraries and tools, organizing the discussion by the various technical areas, as illustrated in Figure 2. We also dedicate discussion to important crosscutting efforts with Spack,† software development kits (SDKs), and the Extreme-Scale Scientific Software Stack (E4S).‡ Subsequent articles in this special issue provide details about the individual library and tools efforts funded by ECP. Related topics are discussed in a special issue of IEEE CiSE (Willenbring et al. 2023; Anzt et al. 2023; McInnes et al. 2023; Heroux 2023; Adamson et al. 2023; Gerber et al. 2023).

Overview

In this section, we introduce the ECP software ecosystem, including E4S, SDKs, and the ECP Software Technology (ST) project (Heroux et al. 2022).

Why a Libraries and Tools Effort

As the field of high-performance computing matures, there is a natural incentive to encapsulate useful functionality into reusable libraries and tools. However, preparations for exascale systems required an increased emphasis on reusability. The critical innovation that enabled exascale systems was the integration of very powerful GPU devices from multiple vendors. These devices rely on massively concurrent execution with complicated memory system architectures where architecture details differed across vendors, as do programming languages and execution environments. Moreover, the increasing complexity of next-generation scientific challenges, which increasingly involve the integration of modeling, simulation, analysis, and learning (McInnes et al. 2021; Draeger and Siegel 2023), demands software ecosystem perspectives, or collections of interdependent products whose development teams have incentives to collaborate to provide aggregate value, where the whole is greater than the sum of its parts.

How Project Scope was Defined and Managed

The scope of the ECP Software Technology Focus Area (sometimes called a 'project' in the following discussion) was developed through collaboration among experts from academia, industry, and government. This collaborative effort ensured that the project stayed relevant to technological advancements and user needs. Managing the scope involved balancing innovation with practical application and adapting to changes as the project progressed.

Planning, Execution, Tracking, and Assessing

The ECP Software Technology Focus Area followed a structured project management approach (Heroux 2023). Clear goals and timelines were set, with specific milestones to track progress. Execution of these plans was closely monitored, with regular assessments to ensure alignment with the initial objectives. This approach helped in dealing with challenges and adjusting the course when necessary.

*https://exascaleproject.org
†https://spack.io
‡https://e4s.io

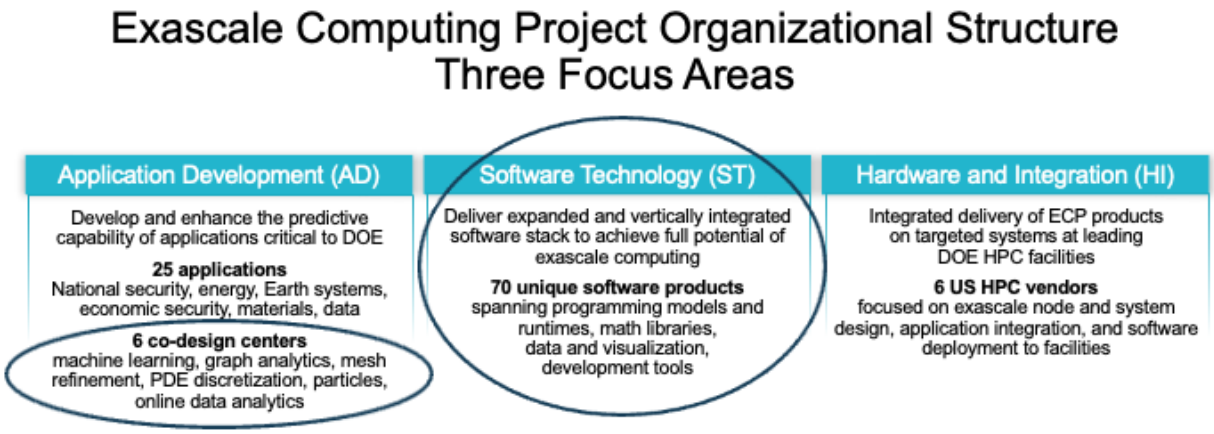

Figure 1. The Exascale Computing Project was composed of three Focus Areas, each with a distinct but complementary responsibility. Application Development (AD) focused on the adaptation of existing application codes and the creation of new ones to produce scientific results on the exascale computing systems at Oak Ridge National Laboratory (the Frontier system), Argonne National Laboratory (the Aurora system), and Lawrence Livermore National Laboratory (the El Capitan system). AD also included efforts in domain-specific suites of reusable libraries and tools in six co-design centers. Software Technologies (ST) focused on developing widely used libraries and tools meant for broad deployment to support applications in their use of the exascale systems. Hardware and Integration (HI) supported vendor efforts and facilitated AD and ST teams' porting efforts to early-access systems and the exascale systems as they were being prepared for deployment. The articles in this special issue come from Software Technology (see Figure 2 for a further breakdown into Technical Areas) and Co-Design teams, circled in the above diagram.

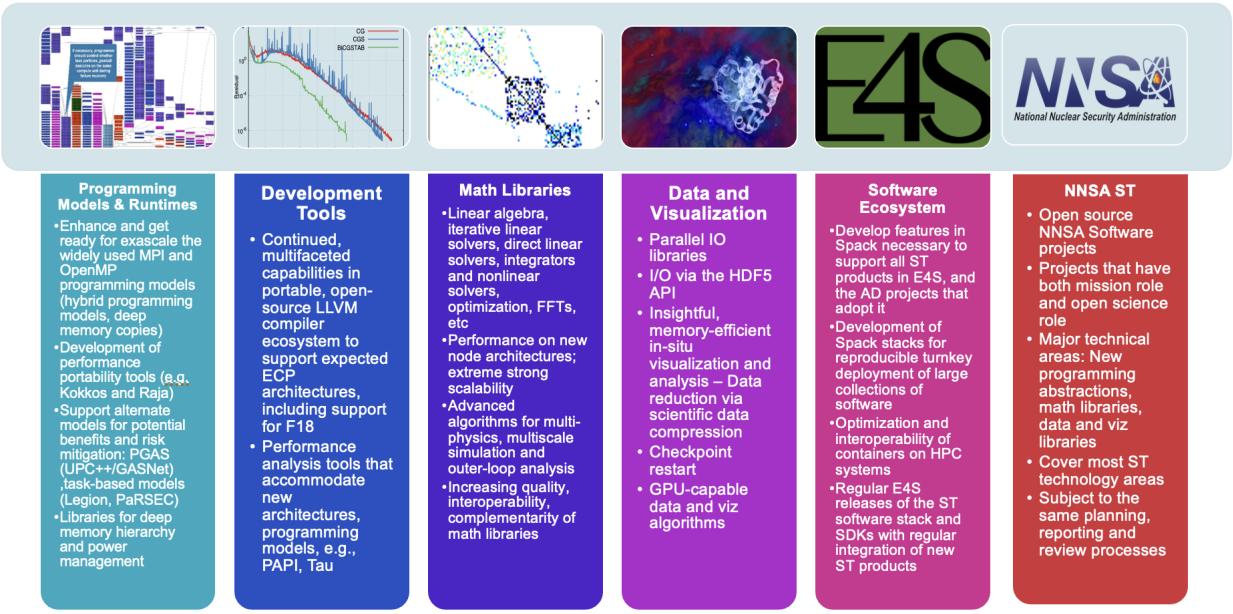

Figure 2. The ECP Software Technology Focus Area was composed of six technical areas. Each area was led by a recognized leader in the community who understood the technical portfolio and the teams. ECP relied on these people to oversee efforts, gather input from the development and user communities, and manage the budget of their technical area. The first five areas were organized by functional scope. The NNSA area was the collection of NNSA-funded efforts that were also focused on open-source development and could benefit from being part of ECP activities and contribute to overall ECP success.

Better Together: Managing a Portfolio

A key decision was whether to develop the libraries and tools as one large project or as several smaller, independent ones. A single large project (i.e., the ECP Software Technology Focus Area, with a portfolio of efforts in complementary technical areas) offered better integration and direction, but this approach also brought challenges in managing a large-scale operation and the associated risks of interdependencies. This hierarchical approach provided the integration essential for progress across the software ecosystem, while also enabling flexibility through smaller efforts to meet specific needs.

Lessons Learned

Several important lessons emerged from the project. The value of collaboration across different fields was clear, as

was the importance of keeping a balance between innovative ideas and their practical implementation. Continuous engagement with end-users was essential to ensure the tools developed were not only advanced but also met the needs of the scientific community. Additional lessons learned regarding ECP overall were documented as part of the project completion, with the goal of informing future large-scale collaborative efforts. Issues from a perspective of ECP applications are discussed in (Draeger and Siegel 2023).

Future Challenges and Next Steps

As work on reusable libraries and tools moves forward, the community faces challenges such as ensuring scalability and efficiency in the face of increasing computational demands. Keeping pace with rapid hardware advancements is another challenge. Addressing these will involve ongoing research and development, incorporating new technologies like artificial intelligence, and advancing dialogue between developers and users. This approach will help libraries and tools efforts continue to evolve so that software ecosystems maintain their key role as a foundation for collaboration and discovery in high-performance computing.

ECP ST Development Subprojects

ECP ST efforts supported development in the following software subprojects spanning five technical areas: (1) Programming Models & Runtimes, see Table 1; (2) Development Tools, see Table 2; (3) Mathematical Libraries, see Table 3; (4) Data & Visualization, see Table 4; and (5) Software Ecosystem & Delivery, see Table 5. Each table lists related products and a link to further information.

The following sections describe the activities and subprojects[§] of each technical area.

Programming Models & Runtimes

The goal of the Programming Models and Runtimes area in ECP was to develop production-ready exascale programming models and runtimes to be used by ECP applications and libraries, co-designed with vendors as appropriate, tuned for application needs and deployed at DOE computing facilities either via the vendor's software stack or as part of E4S.

Background and Context for the Portfolio

The subprojects in this portfolio were selected in 2016 from among the programming models used in applications on DOE HPC systems at that time as well as a few new subprojects. One of the challenges in selecting the portfolio was that it was not known exactly what the architecture of future exascale systems would be or who the vendors of those systems would be. Our best guess was based on roadmaps and architectural trends. At that time, it was generally acknowledged that exascale systems would have powerful multicore nodes. Although GPUs and GPU-based systems existed, it was not known that all exascale systems would have GPUs and would derive most of their computing power from GPUs. Also, the selection of the exascale systems is the result of a competitive procurement process run independently by the DOE computing facilities, the outcome of which is known only at the end of the process when the result is announced. Therefore, deciding which programming models would be needed for unknown systems was challenging.

Some roadmaps in 2016 and earlier suggested that exascale systems could have millions of compute nodes. However, it turned out that the presence of accelerator hardware on compute nodes resulted in individual compute nodes becoming very powerful. As a result, the actual number of compute nodes on the exascale systems is two orders of magnitude lower than the millions expected. Such issues have implications for the programming model and its efficient implementation.

Another challenge was the historical trend of a general reluctance among application communities to embrace new programming models until they have some assurance that the new model will last for a long time, perform well on a variety of platforms, and the effort to port their codes to the new model would be worth it. In other words, there was no guarantee that a new programming model would be adopted by applications.

Programming Models and Runtimes Subprojects

The PMR area included the following subprojects and software products.

1. **MPICH**: This effort extended the widely used MPICH implementation of MPI to run efficiently on exascale architectures. The subproject worked closely with the vendors (HPE and Intel) to enable the vendors to provide tuned MPI implementations based on MPICH on all three exascale systems (Frontier, Aurora, El Capitan). The group also participated in the MPI Forum standardization process on extensions to the MPI Standard for exascale.

2. **OMPI-X**: Since MPI is such a critical component on an HPC system, ECP also supported a second MPI implementation for risk reduction. This effort extended another widely used MPI implementation, Open MPI, to run efficiently on exascale architectures. The group also participated in the MPI Forum standardization process on extensions to the MPI Standard for exascale.

3. **UPC++**: UPC++ provides high-level productivity abstractions for a Partitioned Global Address Space (PGAS) programming model, such as remote memory access, remote procedure call, support for GPUs, and asynchronous mechanisms to hide communication costs.

4. **GASNet-EX**: GASNet-EX is a portable, high-performance communications middleware library that leverages hardware support to implement remote memory access and active message communication primitives. GASNet-EX serves as the communication

[§]Subproject is a specific term that was used within the ECP as the organizational label for the smallest aggregation of individuals as a team developing software capabilities.

Table 1. Programming Models & Runtimes (19 total).

Product	Website
AML	https://github.com/anlsys/aml
Argobots	https://github.com/pmodels/argobots
CAMP	https://github.com/LLNL/camp
CHAI	https://github.com/LLNL/CHAI
GASNet-EX	https://gasnet.lbl.gov
Kokkos	https://github.com/kokkos
Legion	https://legion.stanford.edu
Metall	https://github.com/LLNL/metall
MPICH	https://www.mpich.org
NRM	https://github.com/anlsys/libnrm
Open MPI	https://www.open-mpi.org
PaRSEC	https://icl.utk.edu/parsec
Qthreads	https://github.com/Qthreads
RAJA	https://github.com/LLNL/RAJA
SICM	https://github.com/lanl/SICM
UMap	https://github.com/LLNL/umap
Umpire	https://github.com/LLNL/Umpire
UPC++	https://upcxx.lbl.gov
Variorum	https://github.com/LLNL/variorum

Table 2. Development Tools (23 total).

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

library for programming models such as UPC++, Legion, and Chapel.

5. **Legion**: Legion is a data-centric, task-based programming model for portable parallel programming on distributed, heterogeneous architectures.

6. **PaRSEC**: PaRSEC (Parallel Runtime Scheduler and Execution Controller) is a runtime system and programming toolkit that supports the parallel execution on distributed, heterogeneous systems of programs expressed as a directed acyclic graph of micro-tasks.

7. **Kokkos**: Kokkos is a C++ performance portability abstraction that enables users to write portable code that can run efficiently on heterogeneous node architectures. It uses different backends (CUDA, HIP, SYCL, HPX, OpenMP, C++ threads) to support CPUs and GPUs from multiple vendors. It provides abstractions for both parallel execution of code and data management.

Table 3. Mathematical Libraries (19 total).

Product	Website
AMReX	https://amrex-codes.github.io/amrex/
ArborX	https://github.com/arborx/ArborX
DTK	https://github.com/ORNL-CEES/DataTransferKit
FFTX	https://github.com/spiralgen/fftx
ForTrilinos	https://trilinos.github.io/ForTrilinos
heFFTe	https://icl.utk.edu/fft/
hypre	https://www.llnl.gov/casc/hypre
Kokkoskernels	https://github.com/kokkos/kokkos-kernels
libEnsemble	https://github.com/Libensemble/libensemble
MAGMA	https://bitbucket.org/icl/magma
MFEM	https://mfem.org/
PETSc/TAO	https://www.mcs.anl.gov/petsc
SLATE	https://icl.utk.edu/slate
STRUMPACK	https://portal.nersc.gov/project/sparse/strumpack
SUNDIALS	https://computing.llnl.gov/sundials
SuperLU	https://portal.nersc.gov/project/sparse/superlu
Tasmanian	https://github.com/ORNL/TASMANIAN
Trilinos	https://github.com/trilinos/Trilinos
xSDK	https://xsdk.info

Table 4. Data & Visualization (23 total).

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

Table 5. Software Ecosystem & Delivery (four total).

Product	Website
CharlieCloud	https://github.com/hpc/charliecloud
E4S	https://e4s.io
Flux	https://github.com/flux-framework
Spack	https://github.com/spack/spack

8. **RAJA**: RAJA is a set of C++ libraries providing abstractions for performance portability. It comprises four different libraries: RAJA (kernel execution abstractions), Umpire (memory management interface), CHAI ("smart array" library for automatic data copies between memory spaces), and Camp (C++

metaprogramming facilities focused on HPC compiler compatibility and portability). It also provides multiple backends to run efficiently on a variety of heterogeneous node architectures.

9. **SICM**: The goal of SICM (Simplified Interface to Complex Memory) was to provide an interface for discovering, managing, and sharing data within complex memory hierarchies. It comprises three interrelated components: a low-level API, a high-level API, and a high-level graph interface.

10. **Argo**: Argo provides system software for resource, memory, and power management. It consists of four components: Node Resource Manager (high-level control of node resources), UMap (user-space memory mapped page fault handler for nonvolatile memory), AML (memory library for explicitly managing deep memory architectures), and PowerStack (hierarchical interfaces for power management at the level of batch job schedulers, job-level runtime systems, and node-level managers).

Other programming models, such as OpenMP and OpenACC, were also part of the ECP portfolio and are described in the next section on Development Tools and Compilers.

Lessons Learned

The efforts of the programming models and runtimes area were successful, and the programming models are being used in production by ECP applications. Nonetheless, several notable lessons were learned.

1. The vast majority of ECP application codes still use a distributed-memory programming model using MPI for internode or interprocess communication.

2. Most applications did not have GPU-ready codes at the start of ECP and needed to come up with a strategy for their codes to use GPUs and also GPUs from multiple vendors (AMD, Intel, NVIDIA). This circumstance led to codes using Kokkos, RAJA, OpenMP, or SYCL for GPU programming. Many application teams that previously had codes written in Fortran ported them to C++ to give them a better choice of performance portability tools (Kokkos, RAJA, OpenMP, SYCL).

3. Close collaboration with vendors was very useful; for example in the MPICH subproject, which led to the successful deployment of robust and performant MPI implementations on the exascale platforms.

4. Performance portability across diverse node architectures is important, but there is no convergence yet to any single approach. ECP application codes used a variety of approaches for performance portability, as discussed in (Dubey et al. 2021).

Development Tools and Compilers

The Development Tools (DT) activity of ECP aimed to curate a comprehensive suite of software artifacts designed to enhance developer productivity and optimize performance for exascale computing systems, particularly targeting the architectures of Frontier and Aurora. Central to these efforts was the enhancement of the LLVM compiler ecosystem, which forms the backbone of the programming environment, aiming to improve optimizations, offloading to heterogeneous accelerators, performance profiling, and software correctness. In fact, ECP spearheaded the introduction of the Flang/LLVM for Fortran into the LLVM ecosystem. In addition to core enhancements to the LLVM ecosystem, ECP focused on improving the OpenMP and OpenACC programming models in Clang/LLVM for C/C++ and Flang/LLVM for Fortran, facilitating efficient programming on heterogeneous architectures. In terms of performance tools, ECP supported the TAU performance measurement system, the PAPI hardware counter toolset, and the HPCTookit performance measurement framework. The ECP DT also contributed a long list of benchmarks, V&V suites, and autotuning tools to the broader effort.

Ultimately, these initiatives underscore a strategic approach to tackling the programming, performance, and portability challenges of exascale computing, emphasizing open standards, open software, and robust toolchains for a wide range of architectures and programming models.

Artifacts

ECP DT implemented a number of key software artifacts aimed at enhancing developer productivity and performance efficiency on exascale computing platforms, particularly those architectures anticipated for Frontier and Aurora. Here, we list the prominent artifacts.

1. **LLVM Ecosystem Enhancements**: Improve the core LLVM compiler ecosystem, which is fundamental to the programming environment at exascale. These improvements are intended to enhance optimizations, performance profiling, and correctness aspects of software development.

2. **OpenMP (offload) in Clang/LLVM (Clacc) and Flang/LLVM (Flacc)**: Design and implementation of the OpenMP (heterogeneous) programming model for C/C++ and Fortran in the Clang/LLVM and Flang/LLVM compiler infrastructures, respectively, aiming to provide developers with tools for efficient programming on heterogeneous architectures.

3. **OpenACC in Clang/LLVM (Clacc) and Flang/LLVM (Flacc)**: Design and implementation of the OpenACC heterogeneous programming model for C/C++ and Fortran in the Clang/LLVM and Flang/LLVM compiler infrastructures, respectively, aiming to provide developers with tools for efficient programming on heterogeneous architectures.

4. **Performance Portability and Optimization**: Strategies for leveraging performance modeling and optimization to enable code transformation and improve performance portability across different architectures are highlighted. This includes refining autotuning for OpenMP and OpenACC programming models to directly address the challenges posed by heterogeneous architectures.

5. **TAU Performance Measurement and Analysis Tools**: Improve the TAU performance measurement and analysis tools for target exascale architectures. The aim is to apply these tools to applications to enhance performance understanding and optimization.

6. **HPCToolkit performance analysis tool**: Develops performance analysis tools for optimizing software on extreme-scale parallel systems, focusing on measurement, analysis, and efficiency improvement for exascale computing.

7. **Exa-PAPI performance analysis toolkit**: Enhances the Performance Application Programming Interface (PAPI) for exascale computing environments, focusing on new performance counter monitoring capabilities and power management support for advanced hardware and software technologies.

8. **SYCL Programming Model**: To ensure that implementations of the relatively new SYCL programming model were correct and performant, ECP developed a benchmark suite (HecBench) and evaluated SYCL implementation readiness across relevant architectures.

These artifacts represent a comprehensive suite of tools and enhancements designed to address the programming, performance, and portability challenges anticipated with the advent of exascale computing. The focus on open standards, such as LLVM, OpenMP, OpenACC, and SYCL, alongside efforts in autotuning and performance analysis, underscores a strategic approach to ensuring that the exascale computing ecosystem is robust, efficient, and accessible to developers targeting a variety of architectures and programming models.

Key Challenges

ECP DT overcame several key technical challenges when targeting exascale platforms and applications. These challenges typically revolve around a common set of themes for subprojects aiming to support exascale computing environments. Based on the context of development tools for exascale computing, these challenges included the following.

1. **Heterogeneity**: The heterogeneity in GPU architectures and requisite software posed by the Frontier and Aurora systems was a significant challenge. The architectures were new (to the vendor) and different from other vendors, such as NVIDIA. Many of the hardware capabilities had to be discovered by ECP DT, and then the compilers and tools had to be rewritten to map to that capability.

2. **Scalability**: The massive hierarchical parallelism available in exascale systems and the scalability of associated software systems and applications was a major challenge, far beyond our earlier systems, such as Summit.

3. **Debugging and Profiling at Scale**: Likewise, profiling and debugging at scale takes on an entirely new challenge when targeting tens of thousands of GPUs. DT enhanced several scalable tools for debugging and performance profiling that can handle the complexity and size of exascale applications; however, even then, it often required post-processing terabytes of data to understand the performance analysis.

4. **Programming Model Support**: The number of programming models grew during ECP. For example, the Aurora system added SYCL, which was not a major requirement earlier in the project. Also, performance-portability layers, such as Kokkos, grew in popularity. Extending and optimizing compilers, runtime systems, and libraries to support these emerging programming models while maintaining established programming models became difficult to accomplish with limited resources.

5. **Software Dependencies and Interoperability**: Although tools, such as CMake and Spack, helped to manage the complexity of software dependencies, interoperability between different tools and libraries remained a significant challenge. In particular, device drivers and runtime systems can interfere or conflict with each other during runtime. These dynamic situations can be very difficult to debug because tools have limited visibility into core runtime systems, and they are oftentimes proprietary.

These challenges reflect the complexity of developing and deploying software tools that are critical for the success of exascale computing initiatives. The teams working within the Development Tools section created solutions that addressed many of these issues, enabling the broader scientific community to leverage exascale computing resources effectively.

Collaboration and Deployment

The deployment and sharing of software artifacts within the Development Tools section were strategically executed to ensure widespread adoption and integration into the broader HPC and exascale computing community. A cornerstone of this approach was the collaboration with the LLVM community, a key partnership aimed at enhancing the compiler infrastructure critical for exascale computing. By contributing improvements directly to the upstream LLVM main repository, the subproject teams ensured that enhancements made for exascale computing could benefit a wide range of users beyond the subproject's immediate scope. This collaborative effort underscored a commitment to open-source development and the sharing of advancements with a global developer community.

In addition to collaboration within the LLVM ecosystem, the Extreme-Scale Scientific Software Stack (E4S) played a pivotal role in disseminating these tools, providing a unified platform for easy access to a suite of HPC software. The use of container technologies and the Spack package manager further streamlined the deployment process, facilitating easy installation and management of software dependencies.

Community engagement was further enhanced through workshops, tutorials, and extensive documentation, fostering a collaborative ecosystem for fast knowledge exchange. For example, over the past five years, ECP DT was a significant contributor to the LLVM Annual Development meetings,

where hundreds of researchers and engineers assemble to share progress on the LLVM ecosystem. Continuous integration and testing ensured the tools remained reliable and up-to-date, addressing the evolving needs of the exascale community. This multifaceted strategy, combining direct contributions to LLVM, leveraging E4S, and engaging with the community, significantly amplified the impact and accessibility of the development tools for exascale computing for the long term.

Math Libraries

The Mathematical Libraries effort ensured the healthy functionality of the numerical software on which the ECP applications depend. These libraries span the range from lightweight collections of subroutines with simple APIs to more end-to-end integrated environments. They provide access to a large collection of algorithms ranging from linear/nonlinear solvers, integrators, mathematical optimization, and FFTs to advanced algorithms in multiphysics simulations and outer-loop analysis. The math libraries portfolio contained 16 products, with their functionalities summarized below.

1. **ArborX**: ArborX provides performance portable algorithms for geometric search.
2. **DTK**: The Data Transfer Kit transfers computed solutions between grids with differing layouts on parallel accelerated architectures.
3. **ForTrilinos**: ForTrilinos provides a seamless pathway to generate Fortran library modules from existing C and C++ libraries via a Fortran-targeted extension to the SWIG (Simplified Wrapper and Interface Generator) tool.
4. **heFFTe**: Highly Efficient FFTs for Exascale library provides fast and robust 2-D and 3-D FFTs that target large-scale heterogeneous systems with multicore processors and hardware accelerators.
5. **hypre**: hypre provides scalable linear solvers featuring parallel multigrid methods for both structured and unstructured grid problems.
6. **Kokkos-Kernels**: Kokkos-Kernels implements local computational kernels for linear algebra and graph operations, using the Kokkos shared-memory parallel programming model. It is part of the Kokkos ecosystem offering performance portability.
7. **libEnsemble**: libEnsemble is a Python library to coordinate the concurrent evaluation of dynamic ensembles of calculations. It uses massively parallel resources to accelerate the solution of design, decision, and inference problems and to expand the class of problems that can benefit from increased concurrency levels.
8. **MAGMA**: Matrix Algebra on GPU and Multicore Architectures is a collection of next-generation linear algebra libraries for heterogeneous computing, supporting interfaces for current linear algebra packages and standards (e.g., LAPACK and BLAS).
9. **PETSc/TAO**: The Portable Extensible Toolkit for Scientific Computations/Toolkit for Advanced Optimization provides efficient mathematical libraries for sparse linear and nonlinear systems of equations, time integration, parallel discretization, and numerical optimization.
10. **SLATE**: Software for Linear Algebra Targeting Exascale is a modern replacement of the ScaLAPACK library to provide scalable dense linear algebra capabilities.
11. **STRUMPACK**: STRUctured Matrix PACKage provides linear algebra routines and linear system solvers for dense rank-structured linear systems, as well as sparse direct solvers and preconditioners via low-rank embedding.
12. **SUNDIALS**: SUNDIALS is a SUite of Nonlinear and DIfferential/ALgebraic equation Solvers.
13. **SuperLU**: Supernodal LU is a general-purpose library for the direct solution of sparse, nonsymmetric systems of linear equations.
14. **Tasmanian**: The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN constructs efficient surrogate models for high-dimensional problems and performs parameter calibration and optimization geared towards applications in uncertainty quantification (UQ).
15. **Trilinos**: Trilinos is a collection of reusable scientific software libraries, known in particular for linear solvers, nonlinear solvers, transient solvers, optimization solvers, and uncertainty quantification (UQ) solvers.
16. **xSDK**: Extreme-scale Scientific Software Development Kit.

Collaboration and Deployment

The aforementioned libraries represent a diverse and complementary collection of mathematical software. A multiphysics simulation code often needs to use multiple libraries at the same time. Therefore, it is essential for these libraries to be compiled at the same time and callable between each other. xSDK (Bartlett et al. 2017) becomes the critical bridge to facilitate communication and collaboration among the developers across multiple DOE Labs and universities. xSDK's community efforts have implemented a set of standards for software quality and interoperability deployed a federated Continuous Integration (CI) infrastructure that allows for rigorous software testing on various hardware architectures, and take the challenge of defining software packages that contain compatible versions of the component libraries. xSDK pioneered a set of key elements that address the shortcomings from the past, including community policies, interoperability, common installation via Spack package manager, continuous integration testing, and performance autotuner.

The xSDK subproject held biweekly virtual meetings that included most of the developers of the 15 products. Multiple

teams often held face-to-face meetings at various scientific conferences. The subproject teams made use of collaborative platforms to keep track of progress, brainstorm algorithm ideas, and coauthor papers.

Development Strategy

Prior to ECP, many of the above math libraries existed and provided core math capabilities to DOE's scientific codes. However, a lot of the algorithms and implementations were not nearly as performant as required by the exascale machines. The new developments in ECP were primarily architecture-driven or application-driven.

In recent years, a new hardware trend has been to employ low-precision, special-function units tailored to the demands of AI workloads. Lower-precision floating-point arithmetic can be done several times faster than higher-precision counterparts. Given this architecture feature, the ECP math community created a new multi-precision effort in 2020 to design and develop new numerical algorithms that can exploit the speed provided by lower-precision hardware while maintaining a sufficient level of accuracy that is required by numerical modeling and simulations. This effort resulted in new mixed-precision direct solvers in MAGMA, SLATE, and SuperLU, mixed precision iterative solvers in Ginkgo and Kokkos-Kernels, mixed precision integrators in SUNDIALS, and mixed precision FFTs in heFFTe.

Over the course of interactions with the ECP application teams, the need for batched sparse linear algebra functions emerged in order to make better use of the thousand-way parallelism offered by GPUs when many small algebraic systems are to be solved. These batched systems arise from combustion modeling, bio-chemical modeling, fusion plasma modeling, and nuclear physics simulations, to name a few. The naïve strategy of solving the batched system one by one severely underutilizes the GPU resources, and the overhead of launching many individual kernels with small operations dominates the total runtime.

The ECP math library community seized this opportunity and started developing both batched sparse iterative solvers and batched direct solvers. The batched iterative methods implemented in Ginkgo, Kokkos-Kernels, and PETSc have been successfully deployed in hydrodynamics simulations and plasma simulations, among others. The batched direct solvers are also developed in PETSc and SLATE for dense storage, in MAGMA for banded storage, and in STRUMPACK and SuperLU_DIST for general sparse storage. On the other hand, GPU vendors' math libraries have very limited functionality for batched sparse linear algebra. We know of only Nvidia's cuSolverSP supporting batched QR factorization without pivoting.

Technical Challenges and Solutions

The math library teams had to overcome several key challenges in order to meet the exascale requirements. First, traditional sparse-matrix-based techniques for linear, nonlinear, and ODE solvers, as well as optimization algorithms, are memory-bandwidth limited, with low arithmetic intensity. Second, many algorithms require synchronizations across all compute units, such as the inner product, which hinders the scaling of the solvers. Third, we need to support multiple types of GPUs from different vendors and several programming models. The teams developed a number of robust and sustainable solution strategies to address these challenges, as summarized below.

Reducing communication. These include the communication-avoiding dense LU and QR factorizations in SLATE, the communication-avoiding 3D sparse LU, and sparse triangular solutions in SuperLU_DIST.

Reducing synchronization. These include several pipelined Krylov methods in PETSc/TAO and Trilinos and low-synchronization orthogonalization in Trilinos/PEEKS. These algorithms delay the use of the results of inner products and norms and allow overlapping of the reductions and other computations. The accelerator strategies include using fused computational kernels in PETSc and cluster-based preconditioners to reduce the number of kernel launches.

Increasing arithmetic intensity. These include the use of high-order methods in PETSc/TAO.

Compression techniques. heFFTe uses data compression, such as ZFP, to reduce communication. STRUMPACK uses various off-diagonal low-rank compression algorithms to reduce LU factorization complexity both in flops and in memory.

GPU programming abstraction. Kokkos-Kernels exploits the Kokkos potable layer to implement linear algebra and graph algorithms without using vendor-specific GPU programming languages. ArborX also relies on Kokkos to implement high-performance tree-search methods on different GPUs. Ginkgo employs the backend model to embrace portability, where multiple backends are coded for specific GPUs, but the high-level algorithms use portable APIs to access the kernel operations. PETSc/TAO uses a hybrid method—separating the front-end programming model used by the application and the backend implementations.

Lessons Learned and Future Challenges

A large part of the ECP math success story can likely be attributed to the close integration between DOE applied math/CS experts and scientific applications teams, with emphasis on community ecosystem perspectives.

Working closely with industry partners became essential for the math teams as they developed new capabilities for the math libraries. When we initiated the batched sparse linear algebra work, we invited the technical representatives from the three GPU vendors—AMD, NVIDIA, and Intel—to join our biweekly discussions to set the community standards for the APIs of batched banded solvers, batched sparse iterative solvers, and batched sparse direct solvers. The vendors' input on their GPU features and their existing libraries made the sparse API designs portable and sustainable. We expect that the vendors will adopt these standard APIs for their own implementations, similar to the BLAS standard.

Since the inception of ECP, each math team was asked to develop shared milestones with potential consumers (AD subproject teams) of our products. This is an unusual business model compared to a normal research project. Initially, many PIs of math teams felt uneasy with this approach. But, it became clear over time that this approach ensured that every team collaborated closely with the

application teams, developed what was truly needed, and delivered the functionalities on time. Some of the new developments not in the original ECP scope were also initiated by application needs, such as the batched sparse linear algebra and the GPU-resident sparse direct solver in Ginkgo.

Despite the great advances in ECP's math libraries, many more challenges remain on the horizon. Future HPC node architectures will likely be even more heterogeneous, with different kinds of accelerators on a single node. Most of today's math libraries are not designed for this execution model. We will see more scientific AI/ML workloads on HPC systems, whose computing requirements are different than modeling and simulation codes. Lower precision arithmetic is particularly beneficial, which calls for new error analyses for many algorithms. Better algorithms are needed for building trustworthy language models and surrogate models.

To sustain robust and high-performance math libraries beyond ECP, it is essential to invest in developing the workforce pipeline. Despite the emphasis on employing high-level languages with strong abstraction, the future generations of applied mathematicians still need training to be able to understand compilers and operating systems, and even perform architecture-level code optimization.

Data and Visualization

The goal of the Data and Visualization (DAV) area was to develop and improve data management, data services, and visualization software that supports scientific discovery and understanding at the exascale. This was to be achieved in spite of changes in hardware architecture and the size, scale, and complexity of simulation data produced by exascale platforms. A fundamental challenge for the Data and Visualization area is that exascale system concurrency grew by many orders of magnitude, yet system memory and I/O bandwidth, and persistent capacity continued to grow by only a few orders of magnitude. To address this gap, ECP included some of the top data and visualization teams and their software products to meet this challenge. This area has three subareas: 1) data management and storage, 2) data services - which includes checkpoint restart and scientific data compression, and 3) visualization.

All Data and Visualization product teams worked on making their software exascale-ready: 1) by porting to the chosen exascale architectures, whether GPU accelerators or new I/O subsystems such as DAOS, an open source object store, and 2) by testing functionality and scalability by running first on early access and then full-scale systems. The data and visualization area includes the following software products:

1. **HDF5**: HDF5 is a widely used, open source, data management and storage file format and library. HDF5 is portable and is extensible. HDF5 is highly customizable through its Virtual Object Layer (VOL) and Virtual File Driver (VFD) interfaces. As part of the exascale project, HDF5 researchers and developers designed and implemented asynchronous and caching VOL interfaces as well as direct I/O VFD interfaces for GPU accelerators.

2. **ADIOS**: The Adaptable I/O Systems (ADIOS) addresses I/O and data management challenges posed by large-scale computational science applications running on DOE computational resources. Research and development focused on improving modularity, flexibility, and extensibility and extending, tuning, and hardening core services, such as I/O and staging that support Exascale applications, architectures, and technologies.

3. **UnifyFS**: The UnifyFS subproject provides a user-level file system that is implemented across node-local storage. UnifyFS can store bursts of simulation output, such as checkpoints to the high-performance storage hierarchy starting with node-local fast storage such as NVRAM, and provides I/O performance that can be much faster than the parallel file system at large scales.

4. **Darshan**: The Darshan I/O characterization toolset is an instrumentation tool deployed at facilities to capture information on the I/O behavior of applications running at scale on production systems. Darshan data accelerated root cause analysis of I/O performance problems for applications and assisted in correctness debugging. As part of the exascale project, work focused on extending Darshan to new I/O systems and ensuring readiness on upcoming platforms.

5. **Parallel netCDF (PnetCDF)**: PnetCDF is the parallel I/O library extension of the popular netCDF file format. PnetCDF achieves high performance through non-blocking I/O which aggregates multiple read/write requests into a single operation.

6. **VeloC**: The VeloC product provides a high-performance, scalable checkpoint restart library. VeloC provides a client library for applications to capture their state. This library abstracts the storage hierarchy and supports asynchronous checkpointing.

7. **SZ**: SZ is a user-defined, error-bounded lossy compression library for scientific data.

8. **ZFP**: ZFP is a floating point array primitive that reduces data movement costs by using efficient data representations and compressing data with user-defined error bounds. ZFP supports constant-time read/write random access to individual array elements from its compressed representation. ZFP compressed arrays offer applications the ability to store floating-point data in a compressed form that otherwise would not fit in memory.

9. **VTK-m**: The VTK-m product designed and developed a new set of visualization algorithms that support shared-memory parallelism on many-core CPUs and GPUs.

10. **ASCENT**: The ASCENT product is a lightweight in situ infrastructure for visualization and analysis while the scientific application is running. ASCENT supports data reduction early in the processing pipeline immediately after the data is generated.

11. **ALPINE in situ algorithms**: The ALPINE product offers new in situ algorithms for use in ASCENT. In situ algorithms were developed, including topological analysis, Lagrangian vector flow analysis, adaptive sampling, and statistical and optimal viewpoint selection.

12. **ParaView, Catalyst, and Visit**: ParaView, ParaView's in situ library Catalyst, and Visit are production, large-scale visualization products that are available at supercomputing centers worldwide. The ALPINE team updated these products to run at exascale and integrated them with ASCENT and ALPINE in situ algorithms.

Technical Challenges and Solutions

The DAV product teams had to overcome several challenges to support the exascale ecosystem. The most significant challenge for the DAV product teams is the growing gap between our ability to quickly generate scientific results on processors and the speed at which we can save these results. A theme of challenges in the DAV area is the need for fundamental change in DAV workflows for scientific applications. These challenges are:

Visualization and analysis workflow challenges. In the past, when processor and I/O speeds were more balanced, application teams saved many simulation results for post hoc visualization and analysis. In the exascale era, I/O speeds are typically many orders of magnitude slower than processor speeds. As a result, proportionally, simulation I/O budgets are shrinking at a time when unprecedented time and space simulation details are being calculated. Given this challenge, one solution is to move the visualization and analysis so that it runs as part of the simulation code itself, in situ. Visualization and analysis is typically a significant data reduction process, transforming massive simulation data results into images or summarized data. The key challenge now is this transformational process needs to be done automatically. Post hoc visualization and analysis are typically done interactively, guided by a scientist with the goal of understanding their simulation results. In contrast, in situ visualization and analysis run as part of a batch process over a long period of time. In situ algorithms, therefore, are focused on automatically identifying and acting upon features found in the simulation.

Data handling workflow challenges. Typically, scientists have avoided the use of data compression schemes due to concerns about unquantified data loss. To address this concern, two ECP products focused on providing floating point compression with accuracy guarantees. The SZ and ZFP products use a given accuracy bound to compress scientific data as much as possible while still maintaining the given accuracy. Given the significant I/O bottleneck challenge, some scientists have begun to routinely compress their data as part of their output process. This data-driven approach is complementary to the compute-driven study of the creation of multi-precision solvers by the ECP math library teams.

Abstracting the complexity of the data storage hierarchy. To deal with the significant gaps between processor speeds, memory, and storage, additional layers of the data hierarchy have been added. For example, burst buffers, an intermediate storage layer, are designed to quickly receive and temporarily store simulation checkpoints. A supercomputer center may have archival, center-wide storage, burst buffers, and supercomputing platform storage. DAV data management and storage subprojects worked over the course of ECP to simplify and optimize the use of these diverse layers.

Addressing architecture heterogeneity. GPU accelerators are the engine providing massive numbers of floating operations per second. All DAV products were ported to these architectures. For example, the visualization DAV product, VTK-m, needed to run on these accelerators in order to run in situ. The VTK-m product made use of ECP Programming Models and Runtime portability products, including Kokkos and RAJA to run across the heterogeneous GPU exascale architectures.

Collaboration and Deployment

The Exascale Computing Project used a collaborative development software approach. Gathering all DAV products in one portfolio had several positive benefits.

Complex interactions and dependencies between DAV products could be managed for users. The DAV SDK subproject, part of the E4S effort, managed and released these products. DAV products typically depend on many libraries. Managing these dependencies between the independent products allowed users to more easily use DAV products in combination.

A default data management and storage interface for ECP was selected. HDF5 was selected as the default ECP data management and storage solution. Practically, this meant that if ECP applications were interested in a data management and storage solution, they were guided to try HDF5 first. As an experiment in performance portability, each data management and storage product was asked to write a VOL backend for HDF5 using their own product. This allowed for performance comparisons between VOL implementations as well as offering users the ability to switch to another backend for portability.

Integration of DAV products into other DAV products. As a collection of DAV products working with the same framework, it made sense for DAV products to be available to users from other DAV products. For example, data management and storage subprojects such as ADIOS and HDF5 included methods to use the SZ and ZFP compression libraries.

Integration of DAV products into ECP applications. To integrate with as many ECP applications as possible, DAV teams focused on integrating their products with AMREX, an adaptive mesh refinement co-design framework. AMREX provided the underlying framework for many ECP applications, which then, in turn, had access to many DAV products.

Lesson Learned

Despite the significant gap between compute speed and the rate at which data could be accessed or stored, the DAV portfolio was able to successfully meet these challenges through necessary workflow changes, performance portability, and cross-portfolio capability integration. Lessons learned include:

- DAV solutions that were considered radical at the beginning of the project were embraced by the end. A good example of this was the use of floating-point compression libraries. The compression teams worked with ECP Applications to understand their needs and develop libraries that provided guaranteed accuracy and are now in use today.
- Partnerships with industry partners were a key to success. Kitware, HDFGroup, ParaTools, and the exascale supercomputing vendors were invaluable partners in helping us build the exascale software ecosystem. Standards such as the Visualization Toolkit and HDF5 allowed us to build on well-tested and understood industry standards.
- Close collaborations between DAV and scientific application developers were also key to success. Early application adopters of DAV products provided invaluable feedback in the design of data products in data management and storage, visualization, checkpointing, and compression.

Software Ecosystem and Delivery

The Software Ecosystem and Delivery technical area in the Exascale Computing Project was responsible for coordinating the vertical integration of the software products into the thematic software development kits (SDKs) and the SDKs into the Extreme-Scale Scientific Software Stack (E4S) and for the horizontal delivery of the software stack to the applications and facilities for deployment. This area also provided the critical resources and staffing to enable the software technologies to perform continuous integration testing and product releases and engaged with software and system vendors and DOE facilities staff to ensure the coordinated planning and support of software products. Details on the SDKs and E4S are in a later section. In this section, we consider the underlying capabilities provided by the Packaging Technologies subproject and the ExaWorks subproject on scientific workflows.

Components

Building and integrating software for supercomputers is notoriously difficult, and an integration effort for HPC software at this scale is unprecedented. Moreover, the software deployment landscape is changing as containers and supercomputing-capable software package managers, such as Spack, emerge. Spack enables the automation of the builds of ECP software and allows the software to be distributed in new ways, including as binary packages. Containers enable entire application deployments to be packaged into reproducible images, and can accelerate development and continuous integration (CI) workflows. The Packaging Technologies subproject provided Spack packaging assistance for software developers and ECP facilities and developed new capabilities for Spack to enable automated deployments of software at ECP facilities, in containerized environments, and as part of continuous integration. Concurrently, the Supercontainers subproject investigated and developed technologies and best practices that enable containers to be used effectively at ECP facilities. The goal of the Supercontainers effort was to ensure that HPC container runtimes are scalable, interoperable, and integrated into Exascale supercomputing across DOE.

Exascale computers offer transformative capabilities to combine data-driven and learning-based approaches with traditional simulation applications to accelerate scientific discovery and insight. These software combinations and integrations, however, are difficult to achieve due to challenges in coordinating and deploying heterogeneous software components on diverse and massive platforms. The ExaWorks subproject addressed these challenges by co-designing an open Software Development Kit (SDK) consisting of workflow management systems (WMSs) that can be composed and interoperate through common interfaces. The subproject bootstrapped the SDK with an initial set of tools, designed common interfaces, made tools easier to apply to complex science challenges, and applied the SDK to ECP applications. The subproject was community-centered, working with stakeholders, such as users, developers, and facility representatives, to sustainably address workflow requirements at exascale.

Development Strategy

Spack became the *de facto* delivery method for software products building from source in the ECP. Spack packaging assistance for developers and DOE computing facilities was provided, and new capabilities for Spack that enable automated deployments of software at the facilities were developed. Concurrently, technologies were developed and best practices were documented that enable containers to be used effectively at Facilities.

As scientific workflows continue to become more complex, the ExaWorks subproject addressed this complexity by co-designing an open Software Development Kit consisting of workflow management systems that can be composed and interoperate through common interfaces. The subproject provided a well-engineered and scalable SDK that can be leveraged by new and existing workflows. The SDK is available via E4S and was applied to ECP applications to address broad workflow needs

Technical Challenges and Solutions

Historically, building software to run as fast as possible on HPC machines has been a manual process. Users download source code for packages they wish to install, and they build and tune it manually for high-performance machines. Spack has automated much of this process, but it still requires that users build software. Spack needed modifications to enable it to understand complex microarchitecture details, ABI constraints, and runtime details of exascale machines. This subproject enabled binary packaging and developed new technologies that enable the same binary packages to be used within containers or in bare metal deployments on exascale hardware.

The Supercontainer effort faced similar challenges to deploying containers on HPC machines. Container technology most notably enables users to define their own software environments, using all the facilities of the containerized host OS. Users can essentially bring their own software stack to HPC systems, and they can snapshot

an entire application deployment, including dependencies, within a container. Containers also offer the potential for portability between users and machines. The goal of moving an HPC application container from a laptop to a supercomputer with little or no modification is in reach, but there were a number of challenges to overcome before this was possible on exascale machines. Solutions from industry, such as Docker, assume that containers can be built and run with elevated privileges, that containers are isolated from the host network, file system, and GPU hardware, and that binaries within a container are unoptimized and can run on any chip generation from a particular architecture. These go against the multi-user, multi-tenant user environment of most HPC centers and optimized containers may not be portable across systems.

Emerging exascale workflows pose significant challenges to the creation of portable, repeatable, scalable, and performant workflows. These challenges are both technical and non-technical. On the technical side, WMSs are incapable of supporting heterogeneous co-scheduled and high-throughput workflows, and enabling communication between fine-grained tasks in dynamic workflows. On the nontechnical side, the myriad WMSs that exist, the absence of reusable WMS components, and the lack of user guidance when selecting a WMS have led to a disjoint workflows community that tends toward building ad hoc or bespoke solutions rather than adopting and extending existing solutions. Important challenges are outlined below:

Workflows Community. The workflows, applications, and facility communities are disjoint. Efforts are needed to bring these groups together to define, develop, and integrate common workflow components.

Scheduling. Exascale workflows must manage the efficient execution of diverse tasks (e.g., in runtime, resource requirements, single/multi-node) with complex interdependencies on heterogeneous resources.

Scale and Performance. Emerging workflows feature huge ensembles of short-running jobs, which can create millions or even billions of tasks that need to be rapidly scheduled and executed.

Coordination and Communication. Workflows depend on coordination between the workflow and the tasks within it, a need that requires efficient exchange of data following various communication patterns.

Portability. WMSs are tested on a handful of systems and the frequency by which system hardware and software change makes it impossible to guarantee that a workflow will work on a system in the future.

Key Capabilities and Deployment

The Spack subproject supported ST teams by developing portable build recipes and additional metadata for the ECP package ecosystem. The end goal was to provide a packaging solution that can deploy on bare metal, in a container, or be rebuilt for a new machine on demand. Spack bridged the portability divide with portable package recipes; specialized packages can be built per site if needed, or lowest-common-denominator packages can be built for those cases that do not need highly optimized performance. Packages are relocatable and can be used outside their original build environment. Moreover, Spack provides environments that enable a number of software packages to be deployed together either on an HPC system or in a container.

The Supercontainer subproject documented current practice and leveraged existing container runtimes, but also developed new enabling technologies where necessary to allow containers to run on HPC machines. Several HPC container runtimes (e.g., Shifter, Charliecloud, Singularity) already existed, and this diversity enabled wide exploration of the HPC container design space. The Supercontainers subproject worked with the developers of these subprojects to address HPC-specific needs, including container and job launch, resource manager integration, distribution of images at scale, use of system hardware (storage systems, network and MPI libraries, GPUs, and other accelerators), and usability concerns around interfacing between the host and container OS (e.g., bind-mounting required for hardware support).

The subproject documented best practices to help educate new users and developers to the advantages of containers, and to help ensure efficient container utilization on supercomputers. These living documents are updated periodically in response to lessons learned and feedback. In addition, the subproject identified gaps and implemented changes in the three existing runtimes as needed. The subproject also interfaced with the E4S and SDK teams, as well as AD teams interested in containerizing their applications. The subproject worked to enable these teams to deploy reproducible, minimally-sized container images that support multiple AD software ecosystems.

The ExaWorks subproject laid the foundation for an inherently new approach to workflows: it established an SDK by assembling components from existing WMSs and defined new component interfaces. The ExaWorks SDK provides a robust, well-tested, well-documented, and scalable set of tools and components that can be combined to enable diverse teams to produce scalable and portable workflows for a wide range of exascale applications. Importantly, the subproject did not create a new workflow system nor did it aim to replace the many workflow solutions already deployed and used by scientists, but rather it provided a well-engineered and scalable SDK that can be leveraged by new and existing workflows.

The goals of this subproject were to instantiate the ExaWorks SDK, seed the SDK with robust workflow component technologies, explore pairwise integrations between components and define common component interfaces. The subproject also aimed to impact ECP applications and bring together the workflows community, including developers, ECP applications, and DOE compute facility representatives to collectively address workflows challenges.

Lessons Learned and Future Challenges

Software deployments will continue to become more complex, especially when we require optimized builds for the unique and complicated exascale architectures. Keeping dependencies updated and the software tested on these systems using continuous integration will tax the resources at the Facilities. Software testing that includes interoperability and scalability tests will require further resources, both in terms of people to write the tests and the hours to regularly

run them. These put greater emphasis on using and updating Spack as a solution strategy for large collections of software and for using CI infrastructure and resources.

Containers will become more popular and usable as a way to package the entire environment necessary to run an application on the exascale machines, thereby managing some of the complexity of an application deployment. We expect that performance of an application within a container will be nearly as fast as running the application on bare metal. Application build time will be reduced by using the associated build caches.

Workflows will continue to become more complex to complete their science missions, requiring orchestration of many applications and scripts, executed at various scales across many different resource types, and often reliant on machine learning algorithms for guidance. We expect that hardening workflow management systems and building a community centered around robust and scalable components will be foundational for addressing these complexities. Moreover, we expect that container-based scientific workflows will begin to take off as we transition from demonstrations of applications at scale to performing science with them.

Co-Design

Six ECP co-design centers developed libraries, frameworks, and best practices focused on higher-level abstractions common to multiple ECP (and non-ECP) applications, such as particles, meshes, data-centric, graph, and machine-learning motifs. The goals and mid-ECP status of each of these centers were discussed in detail in a previous special issue of the *International Journal of High-Performance Computing Applications* (Germann 2021). Here, we briefly summarize some of the highlights and lessons learned from the ECP co-design portfolio.

Tools and Libraries

The six co-design centers and some of their key products are briefly summarized here; other articles in this issue provide more details.

The **AMReX** center was responsible for constructing and deploying a framework of the same name to support the development of block-structured adaptive mesh refinement algorithms for solving systems of partial differential equations on exascale architectures. Designed as an exascale successor to BoxLib, several ECP applications were dependent upon AMReX from the beginning of ECP, while others adopted it along the way. Further details about AMReX and the recently developed Python binding, pyAMReX, are provided in another article in this issue.

The Center for Efficient Exascale Discretizations (**CEED**) developed next-generation discretization software and algorithms to enable finite element applications to run efficiently on future hardware. The high-order API library libCEED provided portable and efficient operator evaluation and was used as a backend for the MFEM finite element library, described below. At an even lower level, OCCA provides portability through a lightweight library for just-in-time runtime with heterogeneous platforms. OCCA was used in libCEED, directly in MFEM, and in the newly developed NekRS code, a C++ version of the legacy Nek5000 code built upon the highly optimized OCCA (and libParanumal) GPU kernels.

The Center for Particle Applications **CoPA** developed the Cabana library for particle-based simulations, an extension of the Kokkos performance portability layer providing common particle operations such as neighbor list generation, particle redistribution, halo exchange, particle-grid interpolation, and more. The dense and sparse linear solver operations required by many quantum chemistry and molecular dynamics algorithms were provided through the combination of the high-level Parallel, Rapid O(N), and Graph-Based Recursive Electronic Structure Solve (PROGRESS) library and lower-level Basic Matrix Library (BML). Further details about CoPA are provided in Reeve et al. (2024).

The Center for On-line Data Analysis and Reduction **CODAR** addresses the growing need for data reduction and/or analysis operations to run concurrently with data-generating applications rather than in the traditional post-processing mode. The primary technologies that CODAR developed, supported, and utilized for their co-design analyses include:

1. **Cheetah** is a Python-based experiment harness and campaign management system for running co-design experiments to study the effects of online data analysis. Built on top of the ADIOS ecosystem, Cheetah provides a way to run large campaigns of experiments to understand the advantages and tradeoffs of different compression and reduction algorithms run using different orchestration mechanisms: offline, online (with separate executables), or *in situ* (with a single executable), and with different mappings to available hardware.

2. **Savannah** is a runtime framework for Cheetah that translates experiment metadata into scheduler calls for the underlying system and manages the allocated resources for running experiments. Savanna contains definitions for different supercomputers; based on this information about the target machine, Savanna uses the appropriate scheduler interface (e.g., aprun, jsrun, and slurm) and the corresponding scheduler options to launch experiments.

3. **Z-Checker** is a library to characterize the data and check the compression results of lossy compressors. The Z-Checker tool not only evaluates the compression speed and compression ratio, but also (depending on the application) other measures such as entropy, error distribution, power spectrum, and autocorrelation, either in an online or offline mode.

4. **Chimbuko** captures, analyzes, and visualizes performance metrics for complex scientific workflows and relates these metrics to the context of their execution (provenance) on extreme-scale machines. Because trace data can quickly escalate in volume for applications running on multi-node machines, the core of Chimbuko is an in-situ data reduction component that captures trace data from a running application instance (e.g. MPI rank) and applies machine learning

to filter out anomalous function executions. By focusing primarily on performance anomalies, a significant reduction in data volume is achieved while maintaining detailed information regarding those events that impact the application performance.

The **ExaGraph** Co-Design Center was established to design and develop methods and techniques for efficient implementation of key combinatorial (graph) algorithms, which are some of the most challenging algorithmic kernels to parallelize and scale due to their sparse and irregular memory access patterns. For some problems, such as graph matching, edge covering, graph coloring, and graph partitioning, a full (exact) optimization problem is intractable, but efficient approximation or heuristic algorithms have been developed that have greater concurrency and can be implemented efficiently. Several of these algorithms have been deployed in the Zoltan2 toolkit (a successor to the widely used Zoltan toolkit) of parallel algorithms for partitioning, coloring, ordering, and task placement on modern computing architectures. Other graph algorithms involve community detection, widely used in diverse domains including bioinformatics and social network science, and influence maximization, important in either containing epidemic spread or promoting the spread of marketing campaigns. Lastly, other graph algorithms that commonly arise in computational biology and graph learning rely on basic linear algebraic operations on sparse matrices and vectors; ExaGraph has developed improved algorithms and codes based on them, including PASTIS, a distributed many-to-many protein sequence aligner that relies on sparse matrices and scales to thousands of nodes while being comparable with other sequence aligners in terms of accuracy.

Finally, the ECP Co-design Center for Exascale Machine Learning Technologies (**ExaLearn**) targeted machine learning methods that are common across a number of ECP and other scientific machine learning (SciML) use cases. These fell into four classes of learning problems: surrogate models, inverse solvers, control policies, and design strategies. In addition, a number of proxy applications were developed to represent the unique workload of these new types of data science applications, which differ from traditional scientific computing / computational science applications.

Development Strategy

Each of the ECP co-design centers worked closely with their application “customers” to identify needs and priorities and, in turn, with developers of lower-level libraries and tools, such as Kokkos, MAGMA, and ADIOS. In both cases, this was facilitated by having team members shared across teams so that the end-user perspective was always “in the room,” allowing for a continuous assessment and reprioritization of efforts and co-design assessment of tradeoffs between alternative strategies. For frameworks with established user communities and application customers, such as AMReX, this allowed new functionality to be identified and prioritized. For other newly developed libraries, such as Cabana, the requirements and an API were designed in partnership with the future users before any code was written.

Technical Challenges and Solutions

At the onset of ECP, the community envisioned two exascale ”swimlanes”: a many-core successor to Intel’s Knights Landing, and a GPU-accelerated successor to previous NVIDIA-powered machines, programmed using either CUDA or OpenACC. As ECP progressed, and contracts were awarded for Frontier, Aurora, and El Capitan, code development plans based on these two swimlanes had to adapt. Fortunately, the common abstractions provided by the co-design centers and software technology teams largely allowed application codes to rely on these lower-level libraries and frameworks for portability. But this required a significant redirection of effort, and acceleration of plans to port CPU-native algorithms to GPUs (while reducing or dropping many-core CPU efforts), by the co-design and ST teams.

Key Capabilities and Deployment

New functionality has been developed for existing frameworks, such as the addition of particles and embedded boundaries in AMReX. For all frameworks and libraries, new implementations have been developed that are more suitable for the modern and emerging era of hierarchical parallelism, in some cases through entirely new implementations developed from scratch (e.g., AMReX, NekRS), Finally, some new libraries, and frameworks have been designed and developed, most notably CoPA’s Cabana. Several capabilities were even more collaborative and tailored to specific application needs, including several examples from CODAR, CoPA (ExaMiniMD to SNAP performance gains), and ExaLearn.

Lessons Learned and Future Challenges

Larger ECP teams brought related but distinct development efforts into the same subproject, allowing for useful cross-fertilization of ideas. Examples include Nek and MFEM within CEED, and several mostly single-lab efforts brought into larger multi-lab collaborations such as CODAR, ExaGraph, and ExaLearn. As mentioned above, the inclusion of both higher-level application and lower-level library developers in the co-design team enabled a continuous reassessment of priorities and strategies.

A key challenge is retaining this collaboration and momentum in the future. For instance, in maintaining productive cross-lab collaborations in the face of a natural tendency to return to tribal competition between institutions for smaller teams, and sustaining newer technologies such as Cabana which only have a narrow (but deep) user base drawn from the small pool of ECP applications at the moment, yet have the promise of a broader user community if they can survive the valley of death.

NNSA Libraries and Tools

Efforts funded by NNSA were a part of ECP from the beginning. The software products for the NNSA Software Technologies portfolio were largely focused on unique aspects of the NNSA mission and supporting the National Security Application efforts in ECP. However, each of the software products also maintained their code as open source and also worked to support the wider ECP community.

At the beginning of ECP, the NNSA software products were distributed across the topic areas in Software Technologies: Programming Models & Runtimes, Development Tools, Math Libraries, and Data and Visualization. However, in 2019, a management decision was made to create a new area for NNSA software products and manage them separately from the software products that were funded by the DOE Office of Science. The decision was made largely to ease management challenges that arose because the NNSA products were funded by a different source (NNSA) and because the NNSA subprojects focused on the needs of the NNSA National Security Applications instead of the ECP applications. Even though they were managed separately, the software products in this portfolio had a broad impact across all of ECP and the HPC community.

Tools and Libraries in NNSA Software Technologies

Each of the three NNSA laboratories managed a single subproject that contained multiple independent team efforts.

Los Alamos National Laboratory The subproject from Los Alamos National Laboratory pursued efforts in four team efforts: Legion, LLVM, BEE/Charliecloud, and Cinema.

The Legion data-centric programming system provides unique capabilities and fits within LANL's strategy to develop tools and technologies that enable a separation of concerns of computational physicists and computer scientists. The Legion team focused on the development of significant new capabilities within the Legion runtime that are specifically required to support LANL's ATDM applications and on co-design and integration of advanced programming model research and development within FleCSI, a Flexible Computational Science Infrastructure.

LANL's LLVM-focused team effort included Kitsune, which provides a compiler-focused infrastructure for improving various aspects of the exascale programming environment. The Kitsune efforts primarily focused on advanced compiler and tool infrastructure to support the use of a parallel-aware intermediate representation (IR) in the LLVM compiler infrastructure effort. In addition, the team was actively involved in the Flang Fortran front-end that became an official subproject within the overall LLVM infrastructure. These efforts included interactions across ECP as well as with the broader LLVM community and industry.

The BEE/Charliecloud effort created software tools to increase the portability and reproducibility of scientific applications on high-performance and cloud computing platforms. Charliecloud is an unprivileged Linux container runtime. BEE (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.

The Cinema effort developed scalable solutions for data analysis. Cinema is a novel database approach to saving data extracts in situ, which are then available for post hoc interactive exploration. These data extracts can include metadata, parameters, data visualizations, small meshes, and output plots. Cinema workflows enable flexible data analysis using a fraction of the file storage. Cinema example workflows can be downloaded, built, and run quickly enough to be useful in a variety of applications such as continuous integration (CI), prototyping functionality, or testing analyses.

Lawrence Livermore National Laboratory The subproject from Lawrence Livermore National Laboratory included five team efforts: AID, Flux, Spack and Caliper, MFEM, and RAJA/CHAI/Umpire.

The AID (Advanced Infrastructure for Debugging) effort provided advanced debugging, code-correctness, and testing toolset to facilitate reproducing, diagnosing, and fixing bugs within applications. Four tools were part of this effort: STAT, a highly scalable lightweight debugging tool; Archer, a low-overhead OpenMP data race detector; ReMPI and NINJA, a scalable record-and-replay and smart noise injector for MPI; and FLiT and FPChecker, a floating-point correctness checking tool suite.

The Flux effort focused on the Flux next-generation workload management framework. Flux maximizes scientific throughput by scheduling the scientific work requested by HPC users, also known as jobs or workloads. Using hierarchical scheduling and graph-based resource modeling, Flux manages a massive number of processors, memory, GPUs, and other computing system resources, a key requirement for exascale computing and beyond.

An effort focused on improving tools for HPC developers included Spack and Caliper. Spack is an HPC package manager that automates the process of downloading, building, and installing different versions of HPC applications, libraries, and their dependencies. Spack enables complex applications to be assembled from components, lowers barriers to reuse, and allows builds to be reproduced easily. Caliper is a program instrumentation and performance measurement framework. It is designed as a performance analysis toolbox in a library, allowing one to bake performance analysis capabilities directly into applications and activate them at runtime.

The MFEM effort focused on providing high-performance mathematical algorithms and finite element discretizations to next-generation high-order applications. Among the activities in the MFEM effort was the development of unique unstructured adaptive mesh refinement (AMR) algorithms that focus on generality, parallel scalability, and ease of integration in unstructured mesh applications. Another aspect of the work was porting MFEM to exascale platforms that include GPUs by using mathematical algorithms and software implementations that exploit increasing on-node concurrency.

An effort that focused on programming abstractions included the RAJA, CHAI, and Umpire software libraries that enable application and library developers to meet advanced architecture portability challenges. The goals were to enable the writing of 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.

Sandia National Laboratories The Sandia National Laboratories subproject contained four team efforts: Kokkos, Kokkos Kernels, VTK-m, and OS/ONR.

The Kokkos effort focused on the Kokkos programming model and C++ library that enables performance portable on-compute-node parallelism for HPC C++ applications. The Kokkos library implementation consists of a portable application programmer interface (API) and architecture-specific back-ends, including OpenMP, Intel Xeon Phi, and CUDA on NVIDIA GPUs. These back-ends are developed and optimized as new application-requested capabilities are added to Kokkos, back-end programming mechanisms evolve, and architectures change.

The effort for Kokkos Kernels developed software that implements on-node shared memory computational kernels for linear algebra and graph operations, using the Kokkos shared-memory parallel programming model. Kokkos Kernels supports several Kokkos backends for architectures such as Intel CPUs, KNLs, and NVIDIA GPUs. The algorithms and the implementations of the performance-critical kernels in Kokkos Kernels are chosen carefully to match the features of the architectures. This allows applications to utilize high-performance kernels and transfers the burden to Kokkos Kernels developers to maintain them in future architectures.

The VTK-m effort developed 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 effort built up the VTK-m codebase with the necessary visualization algorithm implementations to run across the varied exascale-era platforms.

The OS and On-Node Runtime (OS/ONR) effort focused on the design, implementation, and evaluation of operating system and runtime system (OS/R) interfaces mechanisms, and policies supporting the efficient execution of application codes on next-generation platforms. The effort priorities included lightweight tasking techniques that integrate network communication, interfaces between the runtime and OS for the management of critical resources portable interfaces for managing power and energy, and resource isolation strategies at the operating system level that maintain scalability and performance.

Development Strategy

The primary objective of the NNSA software technology portfolio was to support the development of new NNSA applications that were started just prior to the founding of the ECP under the Advanced Technology Development and Mitigation (ATDM) program element within the NNSA Advanced Simulation and Computing (ASC) Program. These NNSA software products were developed alongside a broader portfolio of ASC products, and followed the procedures for defining the work scope for the broader portfolio. Annually, a high level scope of work was planned out for these products in the ASC Implementation Plan. Detailed work scope for these products was planned using approved processes within the home institution/laboratory.

The software development teams worked with the NNSA National Security Application teams to co-design much of the functionality that was developed in ECP. The applications targeted running on LLNL's El Capitan exascale system, the architecture of which represents a significant departure from what the application teams used in the past. The software product teams provided tools and capabilities to the application developers at all stages of development. For example, early in the development cycle, the application teams worked to integrate program abstractions such as RAJA and Kokkos to shield the applications from changes in the underlying architecture. Additionally, tools such as debuggers and performance tools were critical for understanding problems in early prototypes. Later, the application teams took advantage of other software capabilities such as those for building and testing, code optimization, I/O, and visualization and analysis. Overall, the NNSA applications were successful because of the broad range of software support developed as part of ECP.

Collaboration with the Broader Community

Many of the software products in the NNSA Software Technologies portfolio received complementary funding from the Office of Science. This additional funding allowed the software teams to broaden their scope beyond supporting the NNSA National Security Applications, which had a remarkable impact on the HPC community. For the wider ECP community, there was substantial benefit gained by the adoption of tools developed by NNSA. Key examples of this are the Spack and Kokkos tools, both of which were adopted by a large number of ECP software products and applications. The NNSA tools provided critical capabilities that supported the software products and applications in meeting their ECP metrics. From the NNSA perspective, the benefit was seen in broad, increased participation in development of the software products, relieving NNSA of the burden of being the sole supporter for development and maintenance of these software products.

Lessons Learned

Overall, two key lessons came from the participation of NNSA software product teams in ECP. First, having a focused mission goal of supporting targeted applications on a particular architecture was highly productive and resulted in great outcomes for the application teams, where the applications met their ECP objectives and their overall NNSA mission goals. Additionally, because the new applications were developed in collaboration with the software product teams and the application teams used the abstractions and tools provided in the NNSA software product portfolio, the new applications will be portable to future architectures that are likely to have different characteristics than current platforms. Second, the collaboration between the NNSA and Office of Science teams provided bidirectional benefits. Prior to ECP, the NNSA and Office of Science teams had much less interaction

and many times similar capabilities were developed on each side, resulting in a duplication of effort. The increased collaboration led to combined efforts to address challenges and resulted in better and more resilient software all around.

Spack

Software build and installation has been a challenge for the HPC community for many years (Dubois et al. (2003); Gamblin et al. (2015)), and ECP posed perhaps the largest software integration challenges in the history of DOE. As described earlier, one of ECP's core mission requirements was to create a robust, reliable, and portable software ecosystem. With nearly 100 different rapidly evolving software packages, each with many dependencies, the ECP stack was around 600 packages in total. All of these packages needed to be built together consistently in many different environments. Integrating them by hand was intractable, especially since they were under constant development. Each time any package changed, its dependent packages needed to be rebuilt and potentially re-integrated to account for changes. Manual software integration might have been a much more difficult problem for ECP, had the Spack (Gamblin et al. (2015)) project not been gaining adoption at the beginning of the project.

Spack's Key Capabilities

Spack is a build tool and *package manager* designed to meet the needs of the HPC community. It enables flexible, portable, optimized builds of large software stacks. Spack's flexibility allowed developers working across ECP, on different machines, with different versions of the same software to continuously integrate and test their work. Without this type of automation, managing and ensuring the robustness of ECP's software products would have required much more work. While there were other build automation tools such as EasyBuild (Hoste et al. (2012)), available in the HPC community, these tools did not provide the flexibility needed for all the environments ECP was required to support.

For most communities, it is sufficient for developers to provide portable, unoptimized binaries of their software. In the HPC community, developers build for many different architectures and software environments, and it is of paramount importance that software be optimized for the host machine. Most applications and software packages (even, in some cases, proprietary ones) are distributed as source so that they can be compiled and integrated by the user or by HPC center staff.

Spack has five key capabilities that enabled it to be successful during ECP:

1. **Spec DSL.** Spack implements a domain-specific language (DSL), for specifications, or *specs* of build configurations. Specs enable users to describe specific versions, features, compilers, and dependencies, either to conditionally enable some aspect of a build or to query for builds matching particular criteria.

2. **Combinatorial Package recipes.** While other tools have a single recipe per configuration of a piece of software, Spack has a single recipe per software package, Recipes are declarative and portable so that they can build configurations that have never been built before. Compared to other systems, this was an especially important feature for ECP, where projects were porting to new systems, new CPUs, and new GPUs frequently. The ability to easily specify a new build with the spec syntax, and to have a package understand it enabled developers to port their software rapidly. Other systems only allow users to *reproduce* pre-specified configurations.

3. **Concretization.** Any parameter of a build may be dependent on some other configuration parameter. For example, certain build options, or *variants*, only exist for certain versions of software. Similarly, only certain versions of some packages may be compatible. Spack's spec syntax allows package authors to specify these types of requirements and define a space of valid builds. *Selecting* the best build configuration that satisfies all requirements is still a daunting challenge. In fact, this problem is NP-hard. The Spack team developed a very general dependency solver called the *concretizer*. Its job is to convert an *abstract*, or underspecified spec, to a *concrete* spec, where all options have a selected value. The concretizer started with a simple, greedy algorithm, but as packages grew in complexity during ECP, it had to be fully rewritten to use a fast, complete solver using Answer Set Programming (ASP) (Gamblin et al. (2022); Gebser et al. (2011)), a logic programming paradigm that excels at solving combinatorial problems.

4. **Binary packaging.** As the ECP ecosystem grew, it became intractable to simply recompile every package for each change. Building software can be extremely slow, especially with portability and GPU frameworks making extensive use of C++ templates. Spack, in collaboration with users at Fermilab and CERN, built up features for binary packaging. Ultimately, this would allow teams to build workflows around Spack that could install 10-100x faster than traditional source builds, while still achieving high levels of optimization. Spack's detailed metadata, particularly around specific microarchitecture information (Culpo et al. (2020)), enabled the creation of optimized binaries, and the redesigned concretizer (Gamblin et al. (2022)) ensured that optimized binaries were selected and used in builds.

5. **Continuous Integration.** In addition to the above key features, the Spack team built an extensive continuous integration (CI) system to ensure that even as the community grew and contributions became more rapid, changes were tested before being integrated into the main Spack branch. With help from Kitware, Amazon Web Services, and the University of Oregon, the Spack team built a cloud build system with Kubernetes and GitLab that ensured that pull requests to key packages in the Spack repositories were built and tested before being merged into the mainline. The system has made Spack builds much more reliable over time. The CI system would not be possible

without binary packaging, as the costs of rebuilding all packages on every commit would be too steep. Even with extensive caching, the build system conducts 40,000 to 100,000 builds per week, depending on Spack project activity.

Impact on Stewardship and Advancement

Spack started out in 2013 as a small project at Lawrence Livermore National Laboratory. It was originally intended to simplify the build of HPC performance tools, automate site deployments in LLNL's Livermore Computing Division, and automate builds for ASC simulation code teams. It was released as open source in 2014, and the project was then published in the main track at the Supercomputing 2015 conference (Gamblin et al. (2015)). After the initial publication, Spack started to gain traction in the broader HPC community. By the start of the ECP, it had amassed around 100 contributors and just over 400 package recipes. By 2017, software projects within ECP needed to build on laptops, commodity clusters, hardware testbeds at exascale facilities, and pre-exascale systems starting to roll out at the DOE laboratories. Every new environment required extensive effort from ECP staff, and between 2017 and 2018, Spack was adopted as the integration framework for E4S. The first version of E4S, built on Spack, was released in November of 2018.

During ECP, Spack became an exemplar for large community open-source projects in DOE. The project was built with community in mind, including not only the community of *users* but also the community of external project *contributors*. In 2016, the majority of ECP software projects were not using continuous integration or automated testing, and many were not hosted on open community sites like GitHub. Still, more were not hosting public documentation. Spack was not the first project to use any of these tools, but it leveraged them to build confidence among potential users and contributors. Automation, documentation, and openness enabled the Spack project to scale dramatically over the course of ECP. Between 2016 and the end of ECP in 2023, Spack grew from just over 400 packages (less than the entirety of the ECP stack) to over 7,500 package recipes. It also grew from around 100 contributors to over 1,300 contributors in the same time period. Widespread usage led to Spack becoming a nearly de-facto standard for HPC software packaging, which further increased its contributor count and helped the sustainability of the project. At the end of ECP, the Spack project was receiving between 300 and 600 contributions (mostly package updates) per month on GitHub.

While ECP's software stack (E4S) is, on its surface, only around 100 packages, those packages rely on around 500 additional transitive dependencies in the Spack ecosystem. Had the Spack project limited itself to only E4S packages, it would not have been able to leverage these network effects, and it would have been *more* difficult to maintain the ECP software stack. Spack enabled *all* HPC software to be managed as a community ecosystem, something that had been extremely difficult in the days before ECP when there was not a sufficiently expressive packaging system to manage a multi-platform, multi-architecture, multi-GPU, multi-version software stack. Spack will live on as the central package manager for the HPC ecosystem, bolstered significantly by the rapid advances made during ECP.

Lessons Learned

We learned many lessons from building out the Spack community during ECP, but the main ones were:

1. **Be technically ambitious**. Many packaging efforts before Spack sought to down-scope the problem of managing combinatorial software builds. They failed to get significant adoption in HPC because of this oversimplification. In HPC, developers *require* options and flexibility for porting and tuning. "Make it as simple as possible, but not simpler" is, sadly, for exascale computing, still extremely complex.

2. **Build communities around packages.** The Spack project was ultimately created for its users, to enable them to build their own packages and projects on top of the core tool. Building a platform like this that increased activity ultimately enabled us to get external users invested in the project, and many ultimately became contributors. ECP itself is part of a broader HPC and open-source community, and it cannot be sustained without keeping *all* of its dependencies up to date.

3. **Invest in automation.** Spack started with automated testing to ensure that external contributions to the core tool were correct, and ultimately the project had to expand its build farm to keep up with the rate of contribution. Without this, the maintainers of Spack would simply have become overwhelmed with the number of contributions. Automation is key to scaling a small development effort up into a large community project.

Future Challenges

Many technical and sustainability challenges remain for Spack after ECP. On the technical side, Spack will need to model software at an increasingly deep level. In particular, the team will need to model very low-level libraries such as C, C++, and Fortran runtime libraries in order to handle compatibility constraints between compilers. While the new solver developed during ECP can help with this, there are scalability challenges with each new piece of information introduced to the combinatorial solver algorithm. Modeling software correctly and in a way that will not cause a combinatorial explosion in the solver is a large challenge.

In addition to technical challenges, Spack has sustainability challenges ahead. The team will need to continue to ensure that a regular flow of contributors helps to maintain the Spack package ecosystem. Currently, the number of contributors continues to increase over time, but encouraging new contributors to join and onboarding them is a constant effort. Spack is a founding project in the newly established High Performance Software Foundation (HPSF)[¶] within the Linux Foundation, and we hope that this will bring more industry supporters to the project. Ultimately, continued

[¶] https://hpsf.io

adoption and contribution is the only recipe for long-term sustainability for Spack.

E4S and the ECP SDKs

ECP applications were built on software components from a variety of categories, including programming models and runtimes, mathematical libraries, data and visualization packages, and development tools. Along with software ecosystems and scientific workflows, these categories each had a Software Development Kit (SDK) subproject associated with it. The software technologies from all of these categories comprise the Extreme-scale Scientific Software Stack (E4S). E4S, which was started as part of ECP, represents a portfolio-driven effort to collect, test, and deliver the latest in reusable open-source HPC software products, as driven by the common needs of applications. E4S provides HPC users with improved access to these products with preinstalled versions, a Spack build cache, and Spack recipes that can be customized to specific needs. Since the majority of users will likely only require a subset of ECP software, E4S offers the option to pick what they need and turn off anything else. It also provides a tool (`e4s_chain_spack.sh`) to easily chain two Spack instances - one installed in a central directory that is typically not writable by users, and one in a user's home directory. This is typically the case with the containerized deployment of E4S. A user can then customize and install new packages in their home directory while using packages installed in the central directory. E4S also coordinates with HPC facilities to help deploy the software on various machines.[‖]

The individual software products in each SDK are specifically tailored to be interoperable where useful, and to be installable in a common ecosystem. Each SDK relies on its developer community to identify its own specific needs, and the SDK teams find and test ways to use related software together and deliver through E4S. For example, the data and visualization SDK established CI testing to ensure its member packages work with common versions of dependent software, and the math libraries SDK, or xSDK, invested heavily in achieving and maintaining interoperability between its member packages.

The E4S and SDK subprojects made it simpler for ECP applications to access required software dependencies on ECP target platforms and drastically lowered the cost of exploring the use of additional ECP ST software. In addition, the SDK effort decreased the ECP software support burden at the major computing facilities by ensuring the general compatibility of ST packages within a single software environment, provided support for the installation of ST packages on facility machines, communicated common requirements for ST software, and facilitated the set up of CI testing at the Facilities.

E4S has the following key features.

- **The E4S suite is a large and growing effort to build and test a comprehensive scientific software ecosystem.** In November 2018, E4S V0.1 contained 25 ECP products. Two years later, E4S V1.2, the fifth E4S release, contained 67 ECP ST products and numerous additional products needed for a complete software environment. Most recently, E4S V24.02 had grown to 122 products. Eventually, E4S will contain all the open-source products needed for a holistic environment.
- **E4S is not an ECP-specific software suite.** The products in E4S represent a holistic collection of capabilities that contain the ever-growing SDK collections sponsored by the ECP and all additional underlying software required to use ECP ST capabilities.
- **E4S is partitionable.** E4S products are built and tested together by using a tree-based hierarchical build process. Because the entire E4S tree is built and tested, users can build any subtree of interest without building the whole stack (Figure).
- **E4S uses Spack.** The Spackmeta-build tool invokes the native build process of each product, enabling the quick integration of new products, including non-ECP products. E4S packages are available as pre-built Spack binaries in the E4S Build Cache (Figure). Binaries for the major operating systems and architectures are added to the build cache as updates become available in Spack.
- **E4S is available via containers and cloud environments.** In addition to a build-from-source capability using Spack, E4S maintains several container environments (e.g., Docker, Singularity, Shifter, CharlieCloud) that provide the lowest barrier to use. Container distributions dramatically reduce installation costs and provide a ready-made environment for tutorials that leverage E4S capabilities. For example, E4S containers now support custom images for ECP applications, such as WDMapp and Pantheon. E4S is also available via AWS and Google Cloud platforms. (Figure)
- **E4S distribution.** E4S products are available at the E4S website.[**]
- **E4S developer community resources.** Developers interested in participating in E4S can visit the E4S-Project GitHub community.[††]

SDKs have the following attributes.

1. **Domain scope:** Each SDK comprises packages whose capabilities are within a natural functionality domain. Packages within an SDK provide similar capabilities that can enable leveraging of common requirements, design, testing, and similar activities. Packages could have a tight complementarity so that ready composability is valuable to the user.
2. **Interaction models:** This is how packages within an SDK interact with each other. Packages may be compatible, complementary, and/or interoperable. Interoperability includes common data infrastructure, or the seamless integration of other data infrastructures, and

[‖] `https://dashboard.e4s.io`

[**] `https://e4s.io`

[††] `https://github.com/E4S-Project`

access to capabilities from one package for use in another.

3. **Community policies:** These include expectations for how package teams will conduct activities, the services they provide, the software standards they follow, and other practices that can be commonly expected from a package in the SDK.

4. **Meta-build system:** This includes robust tools and processes to build from source, install, and test the SDK with compatible versions of each package. This system sits on top of the existing build, install, and test capabilities for each package.

5. **Coordinated plans:** Development plans for each package will include efforts to improve SDK capabilities and lead to better integration and interoperability.

6. **Community outreach:** Efforts to reach out to the user and client communities will include an explicit focus on SDK as a product suite.

Challenges

1. Ecosystem → development, interoperability, testing

   Many scientific applications rely on a collection of software capabilities that vary from application to application. It is important for these software products to function as an ecosystem. Achieving this interoperability requires software to be tested together and be built in common ways. ECP offers two solutions: the Extreme-scale Science Software Stack (E4S) and the software development kits (SDKs).

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

   The SDK effort facilitated the use of common infrastructure, such as CI testing at the major computing Facilities and the Spack package manager. SDK release and delivery goals will benefit from a common package manager and testing infrastructure, including the E4S initiative to provide prebuilt binaries for a variety of architectures.

   Recognizing the release readiness and broader maturity differences between ECP ST products, the release strategy has been to include only those products ready for a joint release in the E4S releases but also to continue to work with other products in preparing for subsequent release opportunities.

2. Community

   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.

   One opportunity for a large software ecosystem effort such as the ECP ST is to foster increased collaboration, integration, and interoperability among its funded efforts. Part of the ECP ST design is the creation of SDKs. SDKs are collections of related software products, called *packages*, in which coordination across package teams will improve usability and practices and foster community growth among teams that develop similar and complementary capabilities.

   Establishing software communities Self-organizing software communities, including developers and users of related technologies, who deeply understand their own requirements and priorities, are well-positioned to determine effective strategies for collaboration and coordination[15].

   Leveraging involvement in ECP software ecosystem activities as well as experience in defining and deploying best practices in software engineering for computational science, members of the IDEAS-ECP team are devising resources[71] to help teams prepare for and participate in ECP software ecosystems and also to increase trust in computational results.

   • Software Development Kits (SDKs) establish collaborative structures for product communities. The SDK approach grew out of the original IDEAS xSDK[4;66;73;74] to provide cross-team collaboration among math library teams by design. The activities conducted within the SDKs have been effective at accelerating design space exploration and making compatible collections of libraries and tools that benefit users, facilities, and the product development teams themselves. SDKs also establish community software policies[48;72] to advance the quality, usability, and interoperability of related software technologies, while supporting the autonomy of diverse teams that naturally have different drivers and constraints.

3. Software Quality

   In October 2020, Version 1 of the E4S Community Policies was announced. The E4S Community Policies serve as membership criteria for E4S member packages. The E4S Community Policies have their genesis in the xSDK Community Policies and have a similar purpose—to help address sustainability and interoperability challenges within the complex software ecosystem of which ECP ST is a part.

   The process of establishing Version 1 of the E4S Community Policies was a multi-year effort led by the ECP SDK team, including representation from Programming Models and Runtimes, Development Tools, Math Libraries, Data and Vis, and Software Ecosystem and Delivery. This team reviewed the existing xSDK Community Policies and selected those policies that were most generally applicable to all

of ECP ST, and not specific to math libraries. From there, the chosen policies were refined, and gaps were analyzed.

In addition to the original policies, a second list of Future Revision policies was also created. These policies are not currently E4S membership criteria but will be very seriously considered in future versions. In most cases, these policies require further refinement or planning prior to adoption. The topics that these policies address provide information about likely subject areas for E4S policies going forward and are critical to ongoing communication with the E4S community.

In December 2021, during the Software Technologies annual review, a lightweight measurement of product compatibility with the E4S Community Policies took place. The results will help guide efforts to increase the number of products that are compatible with the community policies.

Both E4S and SDKs provide important solutions to government agencies—including the National Oceanic and Atmospheric Administration and NASA—and industry users. By increasing the interoperability between ECP's software technologies and coordinating their delivery, E4S and SDKs have helped increase the availability, quality, and long-term sustainability of HPC software.

Extreme-scale Scientific Software Stack (E4S)[23;33] is a curated stack that incorporates the various topical SDKs (including programming models and runtimes, math libraries, data and visualization libraries, and development tools) and relies on the Spack software management ecosystem[67]. E4S facilitates the combined use of independent software packages by application teams, while also improving transparency and reproducibility of computational results.

The SDK solution strategy involves pursuing interoperability and sustainability goals by grouping ST software subprojects 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.

SDKs provide an aggregation of software products that have complementary or similar attributes. The ECP ST uses SDKs to better ensure product interoperability and compatibility. SDKs are also essential aggregation points for coordinated planning and testing. The new layer of aggregation that SDKs represent is important for improving all aspects of product development and delivery. The communities that will emerge from SDK efforts will lead to better collaboration and higher-quality products. Established community policies will provide a means to grow SDKs beyond the ECP to include any relevant external effort. The meta-build configurations based on Spack will play an important role in managing the complexity of building the ECP ST software stack by providing a new layer in which versioning, consistency, testing, and build options management can be addressed at a mid-scope below the global build of ECP ST products.

Summary and Conclusions

The Exascale Computing Project (ECP) made significant strides in developing and optimizing a comprehensive suite of software libraries and tools tailored for exascale systems. The collaborative efforts of ECP's Software Technology and Co-Design teams have led to the creation of robust, scalable, and portable solutions that leverage the unique capabilities of exascale architectures. By addressing challenges related to concurrent execution, memory management, and heterogeneous computing resources, the ECP has not only advanced high-performance computing but also set a precedent for future developments in this field.

Throughout the project, ECP's structured approach, which emphasized clear goals, timelines, and regular assessments, ensured the alignment of software development with the evolving needs of exascale systems. The project's success is evidenced by the integration of ECP-developed products into the Department of Energy's computing facilities, supporting a wide range of scientific applications and demonstrating significant improvements in performance and scalability.

Several lessons emerged from ECP's efforts, highlighting the value of collaboration across various fields, the importance of balancing innovation with practical implementation, and the need for continuous engagement with end-users and vendors. The project underscored the necessity of a collaborative and adaptive approach to handle the complexities of next-generation HPC systems.

Looking ahead, the focus will be on ensuring that these software libraries and tools continue to scale effectively with the rapid advancements in hardware technology. The integration of artificial intelligence and machine learning, coupled with ongoing research and development, will be essential in enhancing the software ecosystem's interoperability and usability. The insights and advancements documented in this overview, along with detailed discussions in subsequent articles, will serve as a valuable resource for guiding future efforts in exascale computing and beyond.

The ECP has laid a foundation for the future of HPC, demonstrating the important role of scalable, efficient, and adaptable software solutions for exascale systems and all systems–from laptops to supercomputers–that contain GPUs and, in the future, similar accelerated computing devices. The continued stewardship and advancement of these technologies will serve as the foundation of many scientific and technological advancements in the years to come.

Acknowledgement

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. LLNL-JRNL-872481.

References

Adamson R, Bryant P, Montoya D, Neel J, Palmer E, Powell R, Prout R and Upton P (2023) Creating continuous integration infrastructure for software development on DOE HPC systems. *IEEE Comput. Sci. Eng.* 25(6). doi:10.1109/MCSE.2024.3362586.

Alexander F, Almgren A, Bell J, Bhattacharjee A, Chen J, Colella P, Daniel D, DeSlippe J, Diachin L, Draeger E, Dubey A, Dunning T, Evans T, Foster I, Francois M, Germann T, Gordon M, Habib S, Halappanavar M, Hamilton S, Hart W, (Henry) Huang Z, Hungerford A, Kasen D, Kent PRC, Kolev T, Kothe DB, Kronfeld A, Luo Y, Mackenzie P, McCallen D, Messer B, Mniszewski S, Oehmen C, Perazzo A, Perez D, Richards D, Rider WJ, Rieben R, Roche K, Siegel A, Sprague M, Steefel C, Stevens R, Syamlal M, Taylor M, Turner J, Vay JL, Voter AF, Windus TL and Yelick K (2020) Exascale applications: skin in the game. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378(2166): 20190056. doi:10.1098/rsta.2019.0056.

Anzt H, Huebl A and Li XS (2023) Then and now: Improving software portability, productivity, and $100\times$ performance. *IEEE Comput. Sci. Eng.* 25(6). doi:10.1109/MCSE.2024.3387302.

Bartlett R, Demeshko I, Gamblin T, Hammond G, Heroux MA, Johnson J, Klinvex A, Li X, McInnes LC, Moulton JD, Osei-Kuffuor D, Sarich J, Smith B, Willenbring J and Yang UM (2017) xsdk foundations: Toward an extreme-scale scientific software development kit. *Supercomputing Frontiers and Innovations* 4(1): 69–82. doi:10.14529/jsfi170104. URL `https://superfri.org/index.php/superfri/article/view/127`.

Culpo M, Becker G, Gutierrez CEA, Hoste K and Gamblin T (2020) archspec: A library for detecting, labeling, and reasoning about microarchitectures. In: *Containers and New Orchestration Paradigms for Isolated Environments in HPC (CANOPIE-HPC'20)*. Virtual Event, part of Supercomputing 2020.

Draeger EW and Siegel A (2023) Exascale was not inevitable; neither is what comes next. *IEEE Comput. Sci. Eng.* 25(3): 79–83. doi:10.1109/MCSE.2023.3311375.

Dubey A, McInnes LC, Thakur R, Draeger EW, Evans T, Germann TC and Hart WE (2021) Performance portability in the Exascale Computing Project: Exploration through a panel series. *IEEE Comput. Sci. Eng.* 23(5): 46–54. doi:10.1109/MCSE.2021.3098231.

Dubois PF, Epperly T and Kumfert G (2003) Why Johnny can't build portable scientific software. *Computing in Science Engineering* 5(5): 83–88.

Gamblin T, Culpo M, Becker G and Shudler S (2022) Using Answer Set Programming for HPC Dependency Solving. In: *Supercomputing 2022 (SC'22)*. Dallas, Texas. LLNL-CONF-839332.

Gamblin T, LeGendre M, Collette MR, Lee GL, Moody A, de Supinski BR and Futral S (2015) The Spack Package Manager: Bringing Order to HPC Software Chaos. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15. ISBN 9781450337236. doi:10.1145/2807591.2807623.

Gebser M, Kaufmann B, Kaminski R, Ostrowski M, Schaub T and Schneider M (2011) Potassco: The potsdam answer set solving collection. *AI Communications* 24(2): 107–124.

Gerber R, Joo B and Parker S (2023) Deploying optimized scientific and engineering applications on exascale systems. *IEEE Comput. Sci. Eng.* 25(6). doi:10.1109/MCSE.2024.3397209.

Germann TC (2021) Co-design in the Exascale Computing Project. *The International Journal of High Performance Computing Applications* 35(6): 503–507. doi:10.1177/10943420211059380.

Heroux MA (2023) Scalable delivery of scalable libraries and tools: How ECP delivered a software ecosystem for exascale and beyond. *IEEE Comput. Sci. Eng.* 25(6). doi:10.1109/MCSE.2024.3384937.

Heroux MA, McInnes LC, Thakur R, Vetter JS, Li XS, Ahrens J, Munson T, Mohror K, Turton TL and ECP Software Technology teams (2022) ECP Software Technology Capability Assessment Report, V3.0. doi:10.2172/1888898.

Hoste K, Timmerman J, Georges A and Weirdt SD (2012) Easybuild: Building software with ease. In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. pp. 572–582. doi:10.1109/SC.Companion.2012.81.

Kothe D, Lee S and Qualters I (2019) Exascale computing in the United States. *IEEE Comput. Sci. Eng.* 21(1): 17–29. doi:10.1109/MCSE.2018.2875366.

McInnes LC, Heroux MA, Bernholdt DE, Dubey A, Gonsiorowski E, Gupta R, Marques O, Moulton JD, Nam HA, Norris B, Raybourn E and Willenbring, J et al (2023) A cast of thousands: How the IDEAS productivity project has advanced software productivity and sustainability. *IEEE Comput. Sci. Eng.* 25(6). doi:10.1109/MCSE.2024.3383799.

McInnes LC, Heroux MA, Draeger EW, Siegel A, Coghlan S and Antypas K (2021) How community software ecosystems can unlock the potential of exascale computing. *Nature Computational Science* 1: 92–94. doi:10.1038/s43588-021-00033-y.

Reeve ST, Fattebert JL, DeWitt S, Joy D, Seleson P, Slattery S, Scheinberg A, Halver R, Junghans C, Negre CFA, Wall ME, Zhang Y, Niklasson AM, Perez D, Mniszewski SM and Belak J (2024) Co-design for particle applications at exascale. *IEEE Comput. Sci. Eng.* : 1–10doi:10.1109/MCSE.2024.3384052.

Willenbring JM, Shende SS and Gamblin T (2023) Providing a flexible and comprehensive software stack via Spack, E4S, and SDKs. *IEEE Comput. Sci. Eng.* 25(6). doi:10.1109/MCSE.2024.3395016.