Title
Then and Now: Improving Software Portability, Productivity, and 100× Performance

Permalink
https://escholarship.org/uc/item/1274c4hs

Journal
Computing in Science & Engineering, 26(1)

ISSN
1521-9615

Authors
Anzt, Hartwig
Huebl, Axel
Li, Xiaoye S

Publication Date
2024

DOI
10.1109/mcse.2024.3387302

Then and Now: Improving Software Portability, Productivity, and 100× Performance

Hartwig Anzt, University of Tennessee, Knoxville, TN, 37996, USA

Axel Huebl and Xiaoye S. Li, Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA

The U.S. Exascale Computing Project (ECP) has succeeded in preparing applications to run efficiently on the first reported exascale supercomputers in the world. To achieve this, it modernized the whole leadership software stack, from libraries to simulation codes. In this article, we contrast selected leadership software before and after the ECP. We discuss how sustainable research software development for leadership computing can embrace the conversation with the hardware vendors, leadership computing facilities, software community, and domain scientists who are the application developers and integrators of software products. We elaborate on how software needs to take portability as a central design principle and to benefit from interdependent teams; we also demonstrate how moving to programming languages with high momentum, like modern C++, can help improve the sustainability, interoperability, and performance of research software. Finally, we showcase how cross-institutional efforts can enable algorithm advances that are beyond incremental performance optimization.

High-performance computing (HPC) enables innovation for scientists and engineers, across exploration and discovery science, design and optimization, or validation of theories about the fundamental laws of nature. Supercomputers enable largest scale data analysis as well as modeling and simulation to study systems that would be impossible to study at the same level of detail in the real world, e.g., due to the size, complexity, physical danger, or cost involved.

In 2016, the U.S. Exascale Computing Project (ECP)[a] started on its mission to accelerate the delivery of a capable exascale computing ecosystem that delivers 50× the application performance of the leading 20 petafloating point operations per second (petaflops) systems. In 2022, the Frontier supercomputer at the Oak Ridge National Laboratory was the first machine benchmarked to compute the LINPACK benchmark at an execution rate of 1 exaflops—thereby fulfilling the ambitious goal of a 5× performance improvement over Summit.[b] With the sunset of the ECP in December 2023, it is time to look at the software side and compare the capability status before and after the ECP.

For this purpose, we describe several software projects that are rooted in mathematical libraries and the application space, and we investigate their performance improvements and sustainability: the Extreme-Scale Scientific Software Development Kit (xSDK) and its constituent libraries, such as Ginkgo, Software for Linear Algebra Targeting Exascale (SLATE), SuperLU, and the laser–plasma modeling application WarpX.[c] We will describe critical facets of how software development methodologies and interdisciplinary teams have been transformed, leading to improvements in the software itself, and why these advances are essential for next-generation science.

[a] https://www.exascaleproject.org

Digital Object Identifier 10.1109/MCSE.2024.3387302
Date of publication 10 April 2024; date of current version 5 July 2024.

[b] https://www.olcf.ornl.gov/summit/
[c] https://warpx.readthedocs.io

SOFTWARE ENGINEERING: THEN AND NOW

Prior to the ECP, many HPC software stacks used for scientific research in the U.S. Department of Energy (DOE) were developed and grew in response to needs of time-limited domain science projects. There was not much coordination among different software teams and products. For example, multiple libraries could not even be built and linked into a single application, e.g., due to name space issues. The naturally growing software stacks also often did not have a defined software development cycle or quality standards to adhere to. Sometimes, even ad hoc solutions implemented to serve certain requirements became an essential part of a major software ecosystem.

The concept of making software sustainable, productive, and reliable through a defined software development process and a culture of collaborative software engineering became popular (and required) only during the ECP. Among the most successful and impactful measures on the side of mathematical software is xSDK.[d] Its community efforts have implemented a set of standards on software quality and interoperability, deployed a federated continuous integration (CI) infrastructure that allows for rigorous software testing on various hardware architectures, and taken the challenge of defining software packages that contain compatible versions of a plethora of independent but interoperable software libraries. xSDK pioneered a set of key elements that addresses the shortcomings from the past:

- *Community policies*: There is a set of mandatory policies [including topics of configuring, installing, testing, message passing interface (MPI) usage, portability, contact and version information, open source licensing, name spacing, and repository access] that a software package must satisfy to be considered xSDK compatible. Also presented are recommended policies (including public repository access, error handling, freeing system resources, and library dependencies), which are encouraged but not required.
- *Interoperability (see Figure 1)*: This enables a collection of related and complementary software packages to be able to call each other so that they can be used simultaneously to solve a complex problem.
- *Easy installation via the Spack package manager*[e]: The release process uses the Spack pull request process for all xSDK-related changes that go into the Spack package manager.
- *Continued systematic testing*: All xSDK packages go through Spack build test cycles on various commonly used workstations. The testing is also extended to multiple DOE Leadership Computing Facility machines. The Gitlab CI (pipeline) infrastructure is used to perform daily runs of multiple tests on different systems.[f]
- *Performance autotuner GPTune*[g]: Each library in xSDK has tunable parameters that may greatly affect the code performance on the actual machine. GPTune uses Bayesian optimization based on Gaussian process regression to find the best parameter configurations. It supports advanced features, such as multitask learning, transfer learning, multifidelity/objective tuning, and parameter sensitivity analysis.

Since the inception of the ECP, the number of libraries in the xSDK collection has grown to 26. Figure 1 illustrates the dependencies among some of the libraries. As shown in this hierarchy, some libraries at the lower level provide commonly used building blocks that are needed by the higher level math libraries and applications.

As an example, the Ginkgo software stack developed under the ECP currently employs 45 CI pipelines on CPU and GPU architectures from AMD, Intel, and Nvidia and has 91% unit test coverage. Likewise, the WarpX application performs CI tests with code reviews on the three major operating systems (Linux, macOS, and Windows) and deploys to three major CPU (x86, ARM, and PPC) and three major GPU (Nvidia, AMD, and Intel) architectures. Reusing the mathematical software in xSDK, the AMReX library[h] became a central dependency of WarpX for its data structures, communication routines, portability, and third-party solvers.

However, it is not the community agreeing on standards, reuse, and the technical realization of rigorous software testing alone that enabled higher productivity and collaboration across institutions. Equally important is the recognition of research software engineering as a profession, establishing career paths and understanding the culture around it.[i] It is an open research software engineering culture across projects

[d] https://xsdk.info

[e] https://spack.io

[f] https://gitlab.com/xsdk-project/spack-xsdk/-/pipelines

[g] https://gptune.lbl.gov

[h] https://amrex-codes.github.io/amrex/

[i] https://us-rse.org and https://society-rse.org

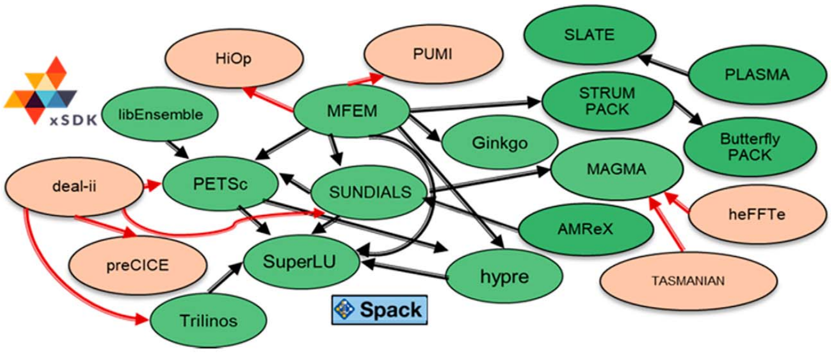


FIGURE 1. xSDK libraries and interoperability tests included in xsdk-examples 0.4.0. Peach ovals represent newly added libraries and red arrows newly featured interoperability. (Courtesy of Ulrike Yang, Satish Balay, and other xSDK developers.) SLATE: Software for Linear Algebra Targeting Exascale; xSDK: Extreme-Scale Scientific Software Development Kit.

and dependencies that drives successful, productive, and resilient software ecosystems, sharing and evolving the practices described in this section.

ALGORITHMS: THEN AND NOW

Exascale machines offer unprecedented degrees of parallelism on the order of tens of millions. This is achieved in the combination of thousands of compute nodes and thousands of compute threads on each node. However, most of the existing algorithms were limited to small to medium degrees of parallelism. Throughout the ECP, we made significant efforts to develop algorithms to better utilize this massively parallel computing power. In this section, we will give several examples to illustrate our algorithm innovations.

To use many compute nodes, an algorithm needs to distribute the data and computing tasks to different nodes, and multiple nodes perform local computation and communicate among each other to finish the entire computation. On exascale machines, the local computation speed is very fast, but communicating a word between two compute nodes is orders of magnitude slower than performing one floating point operation. Therefore, we redesigned many algorithms to reduce the amount of communication.

One example is avoiding communication in the sparse direct linear solver SuperLU.[j] The SuperLU team developed the first communication-avoiding 3-D algorithm framework for a sparse lower-upper (LU) factorization and sparse triangular solution. The algorithm novelty involves a "3-D" process organization and judicious duplication of data between computers to effectively reduce communication by up to several orders of magnitude, depending on the input matrix. The new 3-D code can effectively use 10× more processes than the earlier 2-D algorithm. The sparse LU achieved up to 27× speedup on 24,000 cores of a Cray XC30 [Edison at the National Energy Research Scientific Computing Center (NERSC)]. When combined with the GPU offloading, the new 3-D code achieves up to 24× speedup on 4096 nodes of a Cray XK7 (Titan at the Oak Ridge Leadership Computing Facility) with 32,768 CPU cores and 4096 Nvidia K20x GPUs.[1] The new 3-D sparse triangular solution code outperformed the earlier 2-D code by up to 7.2× when run on 12,000 cores of a Cray XC30 machine. On the Perlmutter GPU machine at NERSC, the new 3-D sparse triangular solution scaled to 256 GPUs, while the earlier 2-D code can only scale up to four GPUs.[2]

An example where the design of a new algorithm class enabled scientific advances is batched iterative solvers. Batched methods are designed to process many problems of small dimension in a data-parallel fashion. They became popular as the hardware parallelism exceeded the problem parallelism, and processing the problems in sequence would be inefficient. Situations where many small systems need to be handled in parallel are common in combustion and plasma simulations but also play a central role in machine learning (ML) methods based on deep neural networks.

Prior to the ECP, batched direct solvers had been developed and used in various applications, but no need for batched iterative methods was formulated. It was the interaction with ECP application specialists that identified the potential of batched iterative methods as approximate solvers for linear problems as part of a nonlinear solver. A cross-institutional task force succeeded in designing batched iterative methods that are performance-portable and suitable for a wide range of applications. The batched iterative functionality

[j] https://portal.nersc.gov/project/sparse/superlu/

deployed in PETSc, Ginkgo, and Kokkos-Kernels is now used in hydrodynamics simulations and plasma simulations, among others. Acknowledging the hardware parallelism exceeding the problem concurrency in many scenarios has resulted in a paradigm change in making algorithms more resource efficient.

While the new batched sparse linear algebra development is driven by application needs, the ECP's new multiprecision algorithm effort is driven by hardware features. In recent years, a new hardware trend has been to employ low-precision, special-function units tailoring to the demands of AI workloads. The lower precision floating point arithmetic can be done several times faster than its higher precision counterparts. In 2020, the ECP community created a new multiprecision effort to design and develop new numerical algorithms that can exploit the speed provided by the lower precision hardware while maintaining a sufficient level of accuracy that is required by numerical modeling and simulations. Examples include the following:

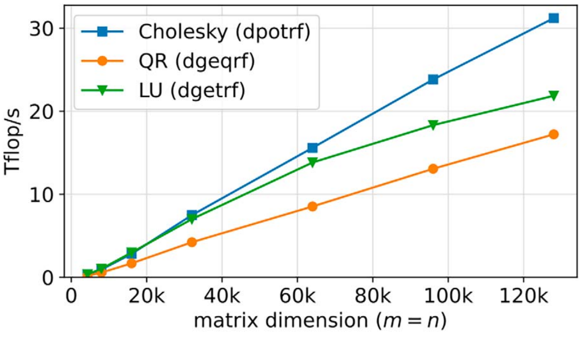
- Mixed-precision iterative refinement for a dense LU factorization in SLATE and a sparse LU factorization in SuperLU achieved 1.8× and 1.5× speedups, respectively.
- Mixed-precision generalized minimum residual method (GMRES) with iterative refinement in Trilinos achieved 1.4× speedup.
- Compressed basis (CB) GMRES in Ginkgo achieved 1.4× speedup, and mixed-precision sparse approximate inverse preconditioners achieved an average speedup of 1.2×.[3]

These speedups coming from the mixed-precision algorithms are "here to stay," as they will carry over to future hardware architectures.

On the domain science level, in applications such as WarpX, algorithmic innovation enabled overcoming traditional limitations in numerical time stepping and improved convergence rates with resolution. Examples include the boosted frame method (using a favorable Lorentz-boosted reference system for simulations); mesh refinement (MR)[4]; advanced pseudo-spectral, GPU-accelerated solvers using GPU-accelerated fast Fourier transforms (FFTs)[5,6]; and accelerated, C++-friendly linear algebra in ECP SLATE.[k]

PERFORMANCE: THEN AND NOW

Through redesign of the algorithms and software implementations, we saw more than 100× gains in speed. In this section, we use the math library SLATE and application code WarpX to illustrate performance differences.



Spock: 4 nodes, 16 AMD MI100 GPUs, 256 AMD EPYC 7662 cores

FIGURE 2. Performance of the GPU routines in SLATE using four nodes of Spock with 16 AMD MI100 GPUs. flop: floating point operation.

Dense linear algebra operations are ubiquitously needed in modeling, simulation, and AI/ML applications. They are also core kernel operations in many sparse computations. For more than two decades, the Scalable Linear Algebra PACKage (ScaLAPACK) library has become the industry standard in distributed memory environments. However, as ScaLAPACK can hardly be retrofitted to support hardware accelerators, the ECP invested in designing a modern replacement—SLATE.

SLATE modernizes the algorithms and software in several ways. It uses a 2-D tiled cyclic data distribution, which enables dynamic scheduling and communication overlapping. It incorporates a number of communication-avoiding algorithms. It relies on standard computation kernels (BLAS++ and LAPACK++) and parallel programming (OpenMP and MPI) for portability. The BLAS++ is an abstraction layer to access multiple GPU back ends: cuBLAS, hip/rocBLAS, and oneAPI.

The speed advantage of SLATE using GPUs over the CPU-based ScaLAPACK code is tremendous. For example, on the pre-exascale system Summit, using 16 nodes with Nvidia V100 and 672 IBM POWER9 cores, SLATE double-precision matrix-matrix multiplication (DGEMM) (dimension 175,000) outperforms ScaLAPACK DGEMM by 65×. Using the same machine configuration, for Cholesky factorization (dimension 250,000), the SLATE code achieved 19× speedup over the ScaLAPACK code. Moving to the exascale systems with the AMD GPUs, Figure 2 shows the GPU-over-CPU speedups of the three routines in SLATE: Cholesky, LU and QR factorizations. These were run on Spock, an earlier system prior to the Frontier exascale machine. Each Spock node consists of a 64-core AMD EPYC 7662 CPU and four AMD MI100 GPUs. The

[k] https://icl.utk.edu/slate/

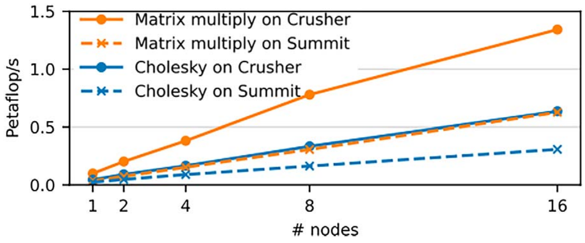


FIGURE 3. SLATE's matrix multiply and Cholesky factorizations scaled and achieved 1.9–2.7× performance improvement per node on Crusher/Frontier nodes with AMD MI250X GPUs compared to Summit nodes with NVIDIA V100 GPUs.

Cholesky factorization achieved 37× speedup, reaching 17% of the machine's peak performance. When the exascale early access system Crusher became available in the early 2023, the SLATE team further improved their codes for the newer AMD GPU, i.e., MI250X.[7] Figure 3 shows up to 2.7× speedup when compared to the Summit Nvidia V100 GPUs.

Performing detailed performance analysis (including roofline modeling of arithmetic intensity) and systematic benchmarking often led to significant rewrites to enable portable performance on new architectures. For instance, the WarpX code already had specially vectorized routines for its core algorithms for Intel Xeon Phi,[l] following the traditional strategy of "porting" per architecture. In the ECP, reimplemented and continuously tuned core routines received an overall 5× improvement, benchmarked in a "figure-of-merit" test (FOM), defined as the weighted particle and cell updates per second, by rewriting them using AMReX's built-in performance portability layer. As an intended side effect, depending on the well-established communication routines in AMReX also ensured scalability and load balancing when running on the scale of full HPC systems. Over the time of the ECP, another 100× of FOM improvement was achieved by combining hardware advances and tuning of co-designed software.[8]

PORTABILITY: THEN AND NOW

Traditionally, high-performance software products were designed for CPU-only supercomputers using MPI for parallelization. It was also common that manually tuned, vectorized code was written for different CPU architectures. Over the last decade, the physical limitations and high power draw of general-purpose CPUs resulted in an increasing number of supercomputers incorporating GPU accelerators. Today, virtually all leadership supercomputers feature accelerators of some form, mostly GPUs from one of the major vendors. Unfortunately, the CPU-centric MPI programming model of the past does not map well to modern GPU-centric supercomputers, and an additional level of parallelism for highly asynchronous execution on modern GPUs is needed. Furthermore, the distinct vendors all prioritize their own programming model over a platform-portable programming language. Finally, GPUs of different generations differ in their compute capabilities and special-function units. This results in a challenge for the software developers to support a wide range of hardware architectures from different vendors and programming models that feature different compute capabilities.

One distinguishes between different levels of portability.[10] The first distinct level is no portability, where the code compiles and runs for only one type of system. Unfortunately, much of the scientific legacy software falls into this category, and adding portability to an existing software stack can be challenging. The next level is partial software portability.[9] An application using such an approach will be dependent on some platform model abstraction. For example, the model could expect any CPU type combined with one or more accelerators, either from AMD or NVIDIA. In such a case, a hybrid programming approach featuring a CPU programming model, like OpenMP, is combined with an accelerator programming model, like HIP, to ensure portability (and possibly good performance) on the machine. As a more advanced case, one might consider full software portability, where the application can execute and run on most platforms, including a reasonable set of hypothetical future machines that might feature field-programmable gate arrays (FPGAs). In this case, a practical example is the SYCL[m] programming model, which features compiler back ends that support some FPGAs, all mainstream HPC accelerators, and ARM-based hardware.

Finally, and especially important for HPC applications, there is the level of performance portability, which means that not only will the code compile and run on target platforms, but it will also achieve high efficiency by providing performance close to the machine's total capabilities. To achieve performance portability, one needs good software design practices (e.g., code portability) *and* full command and understanding of the problems inherent in computing unit granularity versus problem granularity. The latter requires using specific programming techniques to fully express an application's parallelism and scheduling to spread

[l] https://github.com/ECP-WarpX/picsar

[m] https://www.khronos.org/sycl/

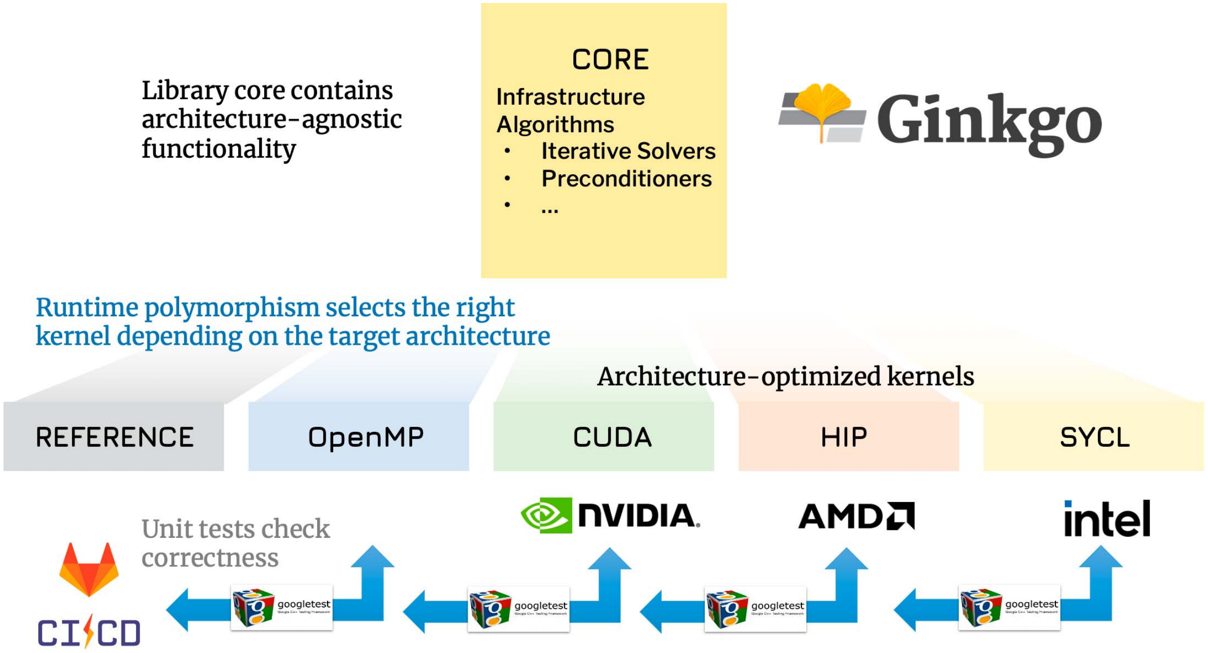

Library core contains architecture-agnostic functionality
CORE
Infrastructure
Algorithms
• Iterative Solvers
• Preconditioners
• ...
Ginkgo
Runtime polymorphism selects the right kernel depending on the target architecture
Architecture-optimized kernels
REFERENCE
OpenMP
CUDA
HIP
SYCL
NVIDIA
AMD
intel
Unit tests check correctness
CI/CD
googletest
googletest
googletest
googletest

FIGURE 4. Overview of the Ginkgo library design using the back-end model for platform portability.[9] High-level algorithms are contained in the library core and composed of algorithm-specific kernels coded for the different hardware back ends.

the workload dynamically, depending on the machine hardware's computing units.

Different strategies exist to tackle the challenge of platform portability. Among the most popular and successful ones are the concept of a *portability layer* and the *back-end model*. The idea behind a portability layer is that the user writes the code once in a high-level language, and the code is then mapped to a source code tailored for a specific architecture and its ecosystem before being executed thanks to an abstraction. Popular examples of portability layers are Kokkos,[n] RAJA,[o] and SYCL.[p] Relying on a portability layer removes the burden of platform portability from the library developers and allows them to focus exclusively on the development of sophisticated algorithms. This convenience comes at the price of a strong dependency on the portability layer, and moving to another programming model or portability layer is usually difficult or even impossible. Furthermore, relying on a portability layer naturally implies that the performance of algorithms and applications is determined by the quality, expressiveness, and hardware-specific optimization of the portability layer.

As an alternative, the idea behind the back-end model is to embrace portability in the software design. The hardware-specific kernels are written separately for the different types of hardware targeted[9]; see Figure 4 visualizing the back-end model used in the Ginkgo library design.[q] Several libraries are using this back-end model effectively, like deal.II[r] and Ginkgo.[9] To use this model, a library must be designed with modularity and extensibility in mind. Only a library design that enforces the separation of concerns between the parallel algorithm and the different hardware back ends can allow for extensibility in the back-end model. The different back ends need to be managed by a specific interface layer between algorithms and kernels. However, the price of the higher performance potential is high: the library developers have to synchronize several hardware back ends; monitor and react to changes in compilers, tools, and build systems; and adopt new hardware back ends and programming models. The effort of maintaining multiple hardware back ends and keeping them synchronized usually results in a significant workload that can easily exceed the developers' resources.[9]

The two strategies for achieving platform portability presented are not necessarily exclusive, and the usage of a hybrid model can be more efficient.[9] One

[n] https://kokkos.github.io
[o] https://raja.readthedocs.io
[p] https://www.khronos.org/sycl/

[q] Ginkgo uses SYCL as one of its back ends, which is a portability layer.
[r] https://www.dealii.org

reason for adopting such a hybrid approach is that not all building blocks are as performance critical or as complex to optimize as others. For those kernels, relying on a performance portability layer allows for reducing the code maintenance and testing complexities as well as focusing on the more performance-critical aspects of the library. One example using the hybrid approach is the PETSc library.[11] The main data objects in PETSc are Vector and Matrix. The PETSc design separates the front-end programming model used by the application and the back-end implementations. Users can access PETSc's Vector, Matrix, and the operations in their preferred programming model, such as Kokkos, RAJA, SYCL, HIP, CUDA, or OpenCL. The back end heavily relies on the GPU vendors' libraries or Kokkos-Kernels[s] to provide higher level solver functions operating on the Vector and Matrix objects.

With the rewrite and advancement from the predecessor Fortran code in Warp to C++ in WarpX, WarpX also benefited from AMReX's performance portability layer, enabling the developers to write most new algorithms in a lambda-based, single-source implementation that supports all targeted architectures. Notably, the performance portability layer in AMReX itself uses a back-end model. Besides portable WarpX performance for HPC, this approach also improved midscale and entry-level user experience: Warp supported only CPU architectures—WarpX runs on major CPU architectures and three different GPU vendors.[8] This enabled WarpX to target all scales of computing that are important for scientific modeling: from laptop to HPC.

Based on analysis of the products in the DOE's software portfolio, platform portability has become a central design principle, thereby increasing the productivity and sustainability of the individual software stacks significantly as well as hardening the resilience of the overall software ecosystem to architectural changes.

LARGE-SCALE SIMULATION CODE: THEN AND NOW

Many scientific problems targeted by application software require excellent weak scaling to the largest available supercomputers. Weak scaling means that, e.g., a 1000× larger problem can be solved in the same time as a 1000× smaller base case if 1000× more theoretical flops are also provided in parallel hardware.

Designing advanced particle accelerators for high-energy physics electron–positron colliders was the primary science driver for WarpX, the advanced electromagnetic particle-in-cell code in the ECP. For this application, an approach relying on subsequent acceleration in stages of laser wakefield accelerators was investigated, an advanced plasma particle acceleration approach that can provide orders of magnitude higher accelerating fields than currently available particle accelerator elements. This and related science drivers in plasma, accelerator, beam, and fusion physics can be simulated in full fidelity with WarpX. Modeling in these domains continues to benefit significantly from scaling and more compute, as it enables, among others, the following:

- *Higher grid resolution*: Modeling of larger systems and higher plasma densities.
- *More particles*: Improved sampling of kinetic, nonequilibrium particle distributions.
- *Includes more microscopic physics*: Better investigation of collisional, quantum, and high field effects.
- *Transition from 2-D to 3-D*: Covering the full geometric effects enables the quantitative prediction of particle energies and the study of particle accelerator stability.
- *Can use long-term stable, advanced solvers*: Modeling of longer physical time scales.

For the laser–plasma physics modeled with WarpX, 3-D domain decomposition is used for multinode parallelism. Besides computation, multiple communication calls between neighboring domains are needed for the time evolution in every simulation step. With its successful scalable implementation, WarpX science runs achieved near-ideal weak scaling over a large variety of CPU and GPU hardware, winning the 2022 ACM Gordon Bell Prize. This included runs on nearly the full scale of Frontier and Fugaku, then brand-new TOP1 and TOP2 in the world.[8]

WarpX improved in several ways over its predecessor Warp and addressed design challenges. Centrally, many of its earlier mentioned advanced algorithms could be implemented and maintained productively, such as MR, because AMReX provided an excellent framework to solve domain decomposition, MR bookkeeping, inherent load balancing, and performance portability. Thus, application developers could focus on implementing a large set of advanced algorithms and optimize their performance.

Beyond the Producer–Consumer Relationship of Scientific Software Development

WarpX development embraced the team-of-teams approach lived in the ECP[12] by depending on co-design centers (like AMReX) and software technology

[s] https://kokkos.org/about/kernels/

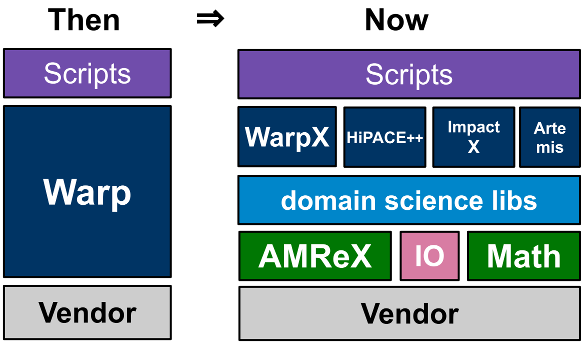


FIGURE 5. Warp to WarpX software stack evolution from 2016 to 2023. Modularization enabled rapid development, performance portability to CPUs and three flavors of GPUs, advanced features, efficient code sharing, and collaboration and spawned off specialized sibling codes in laser–plasma physics (HiPACE++), beam dynamics modeling (ImpactX), and microelectronics (Artemis).

partners for libraries instead of "rolling" their own implementations, as in Warp. Relying on many dependencies did not come without risks since the breakage of a single required dependency on one of many target machines would render WarpX unusable—and, thus, reliability, responsiveness, trust, and often also the possibility to quickly ship a patch to a dependency via open source pull requests are essential to sustain such software relationships. These workflows had to be established, e.g., including upstream testing of WarpX in AMReX changes or providing a ready-to-use developer Docker container for the whole ECP Alpine visualization stack for WarpX CI. Contributions across projects maintained by the dependee and the dependent had to be normalized, forming research software engineering teams with computational physicists, applied mathematicians, and computer scientists.

The dramatic shift in software architecture from Warp to WarpX is shown in a simplified software stack in Figure 5. The box "Math" includes FFT libraries (usually on device from vendors and recently also multidevice, such as heffte/FFTX) and node-local linear algebra for a quasi-cylindrical geometry option (from SLATE's BLAS++ and LAPACK++). The "IO" stack contains the openPMD metadata abstraction[t] on top of parallel HDF5 and ADIOS2; data compressors; and in situ visualization libraries, such as Conduit/Ascent[u] and SENSEI.[v] In greater detail, the Spack dependency tree of WarpX,[w,x] enabling all possible features, includes up to 27 direct dependencies and many transitive packages that are used by the AMReX (1–3), IO (26), and Math (3) boxes. Including all transitive packages and Python dependencies, WarpX currently relies on 101 software packages, according to Spack.

Package management is essential to deploy such a software stack, and so is a modern build system used by package managers. Targeting multiplatform desktop to HPC users, WarpX consequently uses CMake as a platform-agnostic build system and deploys primarily via Spack and Conda, while parts of the system also support PyPI/Pip and Brew.

Software development practices include extensive test coverage;, automation for CI and from-source online documentation (Sphinx, ReadTheDocs, Doxygen, and Breathe); formal code review for all changes, with the approval by at least one maintainer; and a weekly developer meeting. Figure 5 also shows the strategic split of the application layer based on core algorithms and assumptions into libraries and sibling apps, even on the domain science level, to enable easier use and faster development cycles. Code compatibility beyond library sharing is ensured with application programming interface[y] and data standards (openPMD).

In the ECP, WarpX development was able to evolve from earlier short-term projects rooted solely in the domain science application needs, where application R&D had no software sustainability and quality goals on its own, to a strategic multiyear, multidomain HPC effort. This enabled the following:

- *Integration*: Iterations with vendor and ECP libraries over multiple release cycles and deployment through packages to all target platforms.
- *Risk mitigation*: Redesigned input–output with scalable methods from ECP libraries (ADIOS2 and HDF5).
- *Redesigns*: Adjustments were possible, e.g., the following:
  - *Adopting the GPU strategy*: When CPUs and pragma-based compilers lagged behind, WarpX migrated from Fortran to C++.
  - A Python infrastructure overhaul to GPU-capable methods.
  - Performance optimizations led to the redesign of particle data structures in AMReX.

[t] https://www.openPMD.org
[u] https://ascent.readthedocs.io
[v] https://sensei-insitu.org
[w] https://packages.spack.io/package.html?name=warpx
[x] https://packages.spack.io/package.html?name=py-warpx
[y] https://github.com/picmi-standard

- *Trust and collaboration*: Attracted national and international contributors, who trust that their open source contributions will be maintained and enables leveraging of investment in WarpX spin-off/follow-up projects

FUTURE DIRECTIONS

WarpX

WarpX is used as a blueprint to implement compatible, specialized beam plasma and particle accelerator modeling codes,[z] enabling advances in the R&D of particle accelerators, light sources, plasma and fusion devices, astrophysical plasmas, microelectronics, and more. The team continues to address opportunities via novel numerical algorithms, increased massive parallelism, and anticipated novel compute hardware and addresses data challenges with increased in situ processing over traditional postprocessing workflows.

For sustainability, open source development practices will continue to evolve, and WarpX intends to adopt a more formal, open governance model with its national and international partners from national laboratories, academia, and industry.[aa]

Math Software

Building upon the success of the teams' collaboration through the xSDK community platform, the math software developers will continue to innovate the algorithms and software for future architecture and application needs. In particular, the novel algorithms for GPU acceleration paved the way for algorithm design targeting future more heterogeneous architectures with a variety of accelerators. The math teams will expand their algorithm portfolio to meet the needs of AI for science. In addition to new developments, as part of post-ECP work on software stewardship, the math teams will ensure that the libraries developed in the ECP will be sustained for many years to come. This will be achieved through close collaboration and a common governance model across collaborating software stewardship organizations.

ACKNOWLEDGMENTS

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (the Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative. We thank Mark Gates for providing some performance data of SLATE on Summit, a pre-exascale machine. We thank Michael Heroux, Lois McInnes, and Jean-Luc Vay for providing very valuable feedback on the draft manuscripts.

REFERENCES

1. P. Sao, R. Vuduc, and X. Li, "A communication-avoiding 3D algorithm for sparse LU factorization on heterogeneous systems," *J. Parallel Distrib. Comput.*, vol. 131, Sep. 2019, pp. 218–234, doi: 10.1016/j.jpdc.2019.03.004.
2. Y. Liu, N. Ding, P. Sao, S. Williams, and X. S. Li, "Unified communication optimization strategies for sparse triangular solver on CPU and GPU clusters," in *Proc. SC98, High Perform. Netw. Comput. Conf. (SC23)*, Denver, CO, USA, Nov. 13–17 2023, pp. 1–15, doi: 10.1145/3581784.3607092.
3. A. Abdelfattah et al., "*Advances in mixed precision algorithms: 2021 edition*," Lawrence Livermore National Lab., Livermore, CA, USA, Tech. Rep. LLNL-TR-825909 1040257, Aug. 2021. [Online]. Available: https://www.osti.gov/biblio/1814677
4. J.-L. Vay, D. P. Grote, R. H. Cohen, and A. Friedman, "Novel methods in the particle-in-cell accelerator code-framework warp," *Comput. Sci. Discovery*, vol. 5, no. 1, Dec. 2012, Art. no. 014019, doi: 10.1088/1749-4699/5/1/014019.
5. J.-L. Vay et al., "Modeling of a chain of three plasma accelerator stages with the WarpX electromagnetic PIC code on GPUs," *Phys. Plasmas*, vol. 28, no. 2, Feb. 2021, Art. no. 023105, doi: 10.1063/5.0028512.
6. E. Zoni et al., "A hybrid nodal-staggered pseudo-spectral electromagnetic particle-in-cell method with finite-order centering," *Comput. Phys. Commun.*, vol. 279, Oct. 2022, Art. no. 108457, doi: 10.1016/j.cpc.2022.108457.
7. A. YarKhan, M. A. Farhan, D. Sukkari, M. Gates, and J. Dongarra, "*SLATE performance report: Updates to Cholesky and LU factorizations*," ICL, Univ. of Tennessee, Knoxville, TN, USA, Tech. Rep. ICL-UT-20-14, 2020. [Online]. Available: https://icl.utk.edu/files/publications/2020/icl-utk-1418-2020.pdf
8. L. Fedeli et al., "Pushing the Frontier in the design of laser-based electron accelerators with groundbreaking mesh-refined particle-in-cell simulations on exascale-class supercomputers," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2022, pp. 1–12, doi: 10.1109/SC41404.2022.00008.

[z] https://blast.lbl.gov
[aa] https://hpsf.io

9. T. Cojean, Y.-H. M. Tsai, and H. Anzt, “Ginkgo—A math library designed for platform portability,” *Parallel Comput.*, vol. 111, Jul. 2022, Art. no. 102902, doi: 10.1016/j.parco.2022.102902.
10. A. Dubey et al., “Performance portability in the exascale computing project: Exploration through a panel series,” *Comput. Sci. Eng.*, vol. 23, no. 5, pp. 46–54, 2021, doi: 10.1109/MCSE.2021.3098231.
11. R. T. Mills et al., “Toward performance-portable PETSc for GPU-based exascale systems,” *Parallel Comput.*, vol. 108, Dec. 2021, Art. no. 102831, doi: 10.1016/j.parco.2021.102831.
12. E. M. Raybourn, J. D. Moulton, and A. Hungerford, “Scaling productivity and innovation on the path to exascale with a ‘team of teams’ approach,” in *HCI in Business, Government and Organizations. Information Systems and Analytics*, F. F.-H. Nah and K. Siau, Eds., Cham, Switzerland: Springer International Publishing, 2019, pp. 408–421.

HARTWIG ANZT is the is the chair of Computation Mathematics at Technical University Munich, 80333, Munich, Germany, and a research associate professor at the Innovative Computing Laboratory at the University of Tennessee, Knoxville, TN, 37996, USA. His research interests include mixed precision numerical linear algebra, sustainable software, and energy-efficient computing. Anzt received his Ph.D. degree in applied mathematics from the Karlsruhe Institute of Technology. He is a Member of IEEE, GAMM, and SIAM. Contact him at hanzt@icl.utk.edu.

AXEL HUEBL is a computational physicist at Lawrence Berkeley National Laboratory (LBNL), Berkeley, CA, 94720, USA. His research interests include the interface of high-performance computing, laser–plasma physics, and advanced particle accelerator research. Huebl received his Ph.D. degree in physics from Technical University Dresden, Germany. He is a Member of IEEE, APS, and ACM. Contact him at axelhuebl@lbl.gov.

XIAOYE S. LI is a senior scientist at LBNL, Berkeley, CA, 94720, USA. Her research interests include high-performance computing, numerical linear algebra, Bayesian optimization, and scientific machine learning. Li received her Ph.D. degree in computer science from the University of California at Berkeley. She is a fellow of the Society for Industrial and Applied Mathematics and a senior member of the Association for Computing Machinery. Contact her at xsli@lbl.gov.