An Ecosystem of Numerical Libraries to Support and Accelerate Scientific Discoveries

Hartwig Anzt[0000−0003−2177−952X], Satish Balay[0000−0003−4407−205X], Cody J. Balos[0000−0001−9138−0720], David J. Gardner[0000−0002−7993−8282], Pieter Ghysels[0000−0002−5981−5234], Tzanio Kolev[0000−0002−2810−3090], Xiaoye S. Li[0000−0002−0747−698X], Piotr R. Luszczek[0000−0002−0089−6965], Lois Curfman McInnes[0000−0002−6381−4736], Sarah Osborn[0000−0003−3256−3337], Viktor Reshniak[0000−0003−1545−4462], James M. Willenbring[0000−0002−0418−9264], Carol S. Woodward[0000−0002−6502−8659], Ulrike M. Yang[0000−0002−6957−0445]

Abstract The emergence of heterogeneous computers with increasingly complex architectures necessitates continuous adaptation of software to take advantage of increased performance potential. Thus, the use of multiple mathematical libraries designed by expert mathematicians and software developers is crucial for scientific application codes. These libraries are often interconnected, and so, as each library is ported to a new computer architecture, it is also important that these libraries continue to work together. Maintaining such interoperability requires a healthy well-designed software ecosystem with engagement and coordination between library developers who share a common goal of improving software quality, usability, performance portability, and sustainability. This chapter discusses the importance of an ecosystem of high-performance numerical math libraries, including community-driven guidelines to facilitate long-term sustainability and interoperability, and the

S. Balay, L.C. McInnes
Argonne National Laboratory, Lemont, IL, USA, e-mail: {balay,mcinnes}@mcs.anl.gov.

P. Ghysels, X.S. Li
Lawrence Berkeley Laboratory, Berkeley, CA, USA, e-mail: {pghysels,xsli}@lbl.gov.

C.J. Balos, D.J. Gardner, T. Kolev, S. Osborn, C.S. Woodward, U.M. Yang
Lawrence Livermore National Laboratory, Center for Applied Scientific Computing, Livermore, CA, USA, e-mail: {balos1,gardner48,kolev1,osborn9,woodward6,yang11}@llnl.gov.

P.R. Luszczek
MIT Lincoln Laboratory, LLSC, Lexington, MA, USA, e-mail: piotr.luszczek@ll.mit.edu.

V. Reshniak
Oak Ridge National Laboratory, Oak Ridge, TN, USA, e-mail: reshniakv@ornl.gov.

J. Willenbring
Sandia National Laboratories, Albuquerque, NM, USA, e-mail: jmwille@sandia.gov.

H. Anzt
Technical University of Munich, Munich, Germany, e-mail: hartwig.anzt@tum.de.

positive impact of the ecosystem on applications; in particular, the focus is on the Extreme-Scale Scientific Software Development Kit (xSDK)[1].

1 Introduction

Application scientists want to create larger and more detailed simulations of physical phenomena. Achieving these goals required the development of exascale computers, i.e., systems capable of exceeding a quintillion, or 10^{18}, floating point calculations per second. This has led to the rise of ever more complex computer architectures. The increase in computational power and the availability of more memory provided the opportunity for application scientists to solve larger more complex problems with higher fidelity. However, in order to take advantage of the full potential of these high-performance computers, a variety of challenges had to be overcome. Application codes and third-party libraries needed to be adapted to leverage the new computing capabilities, often requiring the introduction of new programming models. More complex science models required new and improved mathematical algorithms to enable the solution of harder problems. These challenges go beyond the ability of a single team of application developers, since they require expertise in a variety of different disciplines, including software development skills for new computer architectures and deep understanding of mathematics in various areas, such as discretization of models on structured and unstructured grids, solution of linear and nonlinear systems, optimization, and so on. Dealing with these issues requires an ecosystem of libraries and tools that can support these application codes.

While the word "ecosystem" usually evokes images of an environment comprised of living organisms, like plants and animals, here it is more generally defined as *a group of independent but interrelated elements that comprise a unified whole*. One example of such an ecosystem that can support applications is E4S,[2] the Extreme-Scale Scientific Software stack [12, 21], which is comprised of packages from a large variety of different software categories, including programming models, runtime systems, development tools, numerical libraries, data and visualization software, and workflows. However, we focus here on a smaller ecosystem, which is also contained within E4S, the Extreme-Scale Scientific Software Development Kit, in short xSDK,[3] which consists of a variety of independently developed numerical libraries [6, 24, 22], including parallel, often GPU-enabled, software for solvers, meshes, discretizations, and more. It provides application developers a variety of the building blocks they need.

An efficient ecosystem requires that all its elements work well together, i.e., it needs to enable an application to smoothly build when using components of the system in combination. Achieving this cohesion presents several challenges,

[1] xSDK: `https://xsdk.info/`

[2] E4S: `https://e4s-project.github.io/`

[3] xSDK: `https://xsdk.info/`

particularly when the individual libraries are managed by independent teams. To facilitate the combined build and use of multiple packages in the kit, the xSDK community developed a set of guidelines, known as the xSDK community policies, which are periodically reevaluated and updated. All xSDK member packages have at least one interoperable connection with another xSDK member, leading to additional challenges, particularly when testing the entire xSDK. A cohesive build of these packages is enabled by the Spack[4] package manager.

The rest of this chapter is organized as follows. We discuss the history of the xSDK in Section 2, the xSDK community policies in Section 3, and delve deeper into interoperability in Section 4, including the introduction of a suite of example codes demonstrating interoperability. Section 5 discusses the release process and testing challenges of the xSDK. Before concluding in Section 8, we highlight some examples of the impact the xSDK has made on applications in Section 6 and the benefits of automatic performance tuning of xSDK libraries in Section 7.

2 History

While there have been several efforts to address challenges inherent in a numerical software ecosystem in the past (e.g., see [11] and references therein), the xSDK was the first to truly deliver on this promise. The xSDK effort began in 2014 as part of the Interoperable Design of Extreme-scale Application Software (IDEAS) project [17].[5] While the original IDEAS project (now dubbed IDEAS-Classic) focused on use case requirements for multiphysics and multiscale terrestrial ecosystem modeling, the compatibility, usability, quality, and process improvements made for important DOE libraries – including *hypre*[6], PETSc[7], SuperLU[8], and Trilinos[9] – were broadly beneficial and laid the groundwork for the expansion of the xSDK.

The first release of the xSDK in 2016 included only the four numerical libraries listed above and the two domain components Alquimia[10] and PFLOTRAN[11]. At the time of the release, this was a significant accomplishment, since previously it was not possible to link the four libraries into a single executable due to incompatibilities. The dependency graph for the IDEAS ecosystem of 2016 is shown in Figure 1. The xSDK libraries are shown in green. Another significant benefit from the early days of the xSDK was breaking down barriers between the development teams of different libraries by establishing compatible goals as well as a common vocabulary.

---

[4] Spack: `https://spack.io/`

[5] `https://ideas-productivity.org/activities/ideas-classic/`

[6] `https://llnl.gov/casc/hypre`

[7] `https://petsc.org`

[8] `http://portal.nersc.gov/project/sparse/superlu/`

[9] `https://trilinos.org/`

[10] `https://ideas-watersheds.github.io/software-ecosystem/codes/alquimia`

[11] `https://pflotran.org`

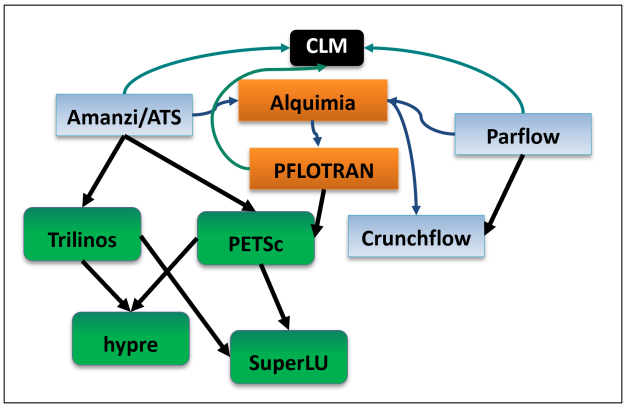

Fig. 1: Application packages (black, blue, and orange) and mathematical libraries (green) needed to generate multiscale, multiphysics integrated surface-subsurface hydrology models.

A tangible outcome of establishing these goals was the creation of the initial xSDK Community Policies, which are discussed in Section 3.

As a result of the early successes under the IDEAS project, the xSDK was funded as its own project as part of the Software Technologies focus area of the Exascale Computing Project (ECP) [14, 8] under the Math Libraries thrust, while the IDEAS-ECP project was funded to help mitigate the challenges of software development in emerging exascale environments and thereby to enhance the productivity and sustainability of the software ecosystem. The ECP brought together the developers of 70 distinct software technologies to create a rich and sustainable scientific software ecosystem [18] that supported a diverse suite of applications in chemistry, materials, energy, Earth and space science, data analytics, optimization, AI, and national security [1]. Projects targeted portable performance across multiple exascale computer architectures. ECPâĂŹs aggressive goals required intensive software refactoring and development [3], involving more than 1,000 researchers across DOE labs and collaborating universities, as well as partnerships with DOE computing facilities, industry, and other US agencies.

During the ECP, xSDK membership grew over the years. It currently includes 26 math libraries: AMReX[12], ArborX[13], ButterFlyPACK[14], DataTransferKit[15],

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

[13] https://github.com/arborx/arborx

[14] https://github.com/liuyangzhuan/ButterflyPACK

[15] https://github.com/ORNL-CEES/DataTransferKit

deal.ii[16], ExaGO[17], Ginkgo[18], heFFTe[19], HiOp[20], *hypre*, libEnsemble[21], MAGMA[22], MFEM[23], Omega_h[24], PETSc/TAO, PHIST[25], PLASMA[26], preCICE[27], PUMI[28], SLATE[29], SLEPc[30], STRUMPACK[31], SUNDIALS[32], SuperLU, TASMANIAN[33], and Trilinos, and 2 application packages: PFLOTRAN[34], and Alquimia[35].

3 xSDK Community Policies

The ecosystem of numerical libraries for scientific applications is a confluence of the developer community with the hardware platforms and software tools they target. To address a range of technical and social challenges that arise when building and sustaining a rich software ecosystem a shared set of policies and practices agreed to by all members was developed. This helps maintain community cohesion while keeping the common goals in clear view of the members. Akin to a founding document, the policies first and foremost indicate that the community is in agreement on core principles which define expectations that must be fulfilled by all participating libraries.

Drawing on decades of open-source library development experience in the U.S. Department of Energy, the xSDK community policies are intended to create a highly productive, sustainable, and interoperable software library ecosystem. The short form of the latest version of the community policies is presented in Figure 2 and a detailed version is available on GitHub.[36] The mandatory policies can be seen

[16] http://www.dealii.org/
[17] https://github.com/pnnl/ExaGO
[18] https://github.com/ginkgo-project/ginkgo
[19] https://bitbucket.org/icl/heffte
[20] https://github.com/LLNL/hiop
[21] https://github.com/Libensemble/libensemble
[22] http://icl.utk.edu/magma/
[23] https://mfem.org/
[24] https://github.com/sandialabs/omega_h
[25] https://bitbucket.org/essex/phist
[26] http://icl.utk.edu/plasma/
[27] https://www.precice.org/
[28] https://github.com/SCOREC/core
[29] https://bitbucket.org/icl/slate
[30] http://slepc.upv.es/
[31] http://portal.nersc.gov/project/sparse/strumpack/
[32] https://computing.llnl.gov/projects/sundials
[33] https://ornl.github.io/TASMANIAN
[34] http://www.pflotran.org/
[35] https://github.com/LBL-EESA/alquimia-dev/
[36] https://github.com/xsdk-project/xsdk-community-policies

as quality assurance for participation in the xSDK ecosystem and is complemented by a set of recommended policies that have been identified to improve software productivity and sustainability but that cannot be fulfilled or do not apply to all software libraries. The policies are regularly reviewed and reevaluated as necessary to stay relevant with current software development and deployment best practices. Revisions or additions to the community policies can be suggested at anytime by opening a GitHub issue for discussion. Changes will then be voted on for inclusion by xSDK members.

Mandatory xSDK policies: must be satisfied

M1. Support portable installation through Spack (includes xSDK Spack variant guildelines)
M2. Provide a comprehensive test suite.
M3. Employ user-provided MPI communicator.
M4. Give best effort at portability to key architectures.
M5. Provide a documented, reliable way to contact the development team.
M6. Respect system resources and settings made by other previously called packages.
M7. Come with an open-source license.
M8. Provide a runtime API to return the current version number of the software.
M9. Use a limited and well-defined symbol, macro, library, and include file name space.
M10. Provide publicly available repository.
M11. Have no hardwired print or IO statements.
M12. Allow installing, building, and linking against an outside copy of external software.
M13. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.
M14. Be buildable using 64-bit pointers. 32 bit is optional.
M15. All xSDK compatibility changes should be sustainable.
M16. Have a debug build option.
M17. Provide sufficient documentation to support use and further development.

Recommended xSDK policies: encouraged, but not required

R1. Provide at least one validation test that can be invoked through Spack**.**
R2. Possible to run test suite under valgrind in order to test for memory corruption issues.
R3. Adopt and document consistent system for error conditions/exceptions.
R4. Free all system resources it has acquired as soon as they are no longer needed.
R5. Provide a mechanism to export ordered list of library dependencies.
R6. Provide versions of dependencies.
R7. Have README, SUPPORT, LICENSE, and CHANGELOG file in top directory.
R8. Provide version comparison preprocessor macros.

Fig. 2: xSDK community policies version 1.0.0

4 Interoperability

For a software package to become an xSDK member it must comply with the mandatory community policies and have interoperability with at least one other xSDK member package. There are different levels of interoperability:

- Packages are used side by side in an application.
- Libraries exchange (or control) data with each other
- A library calls another library to perform a unique computation.

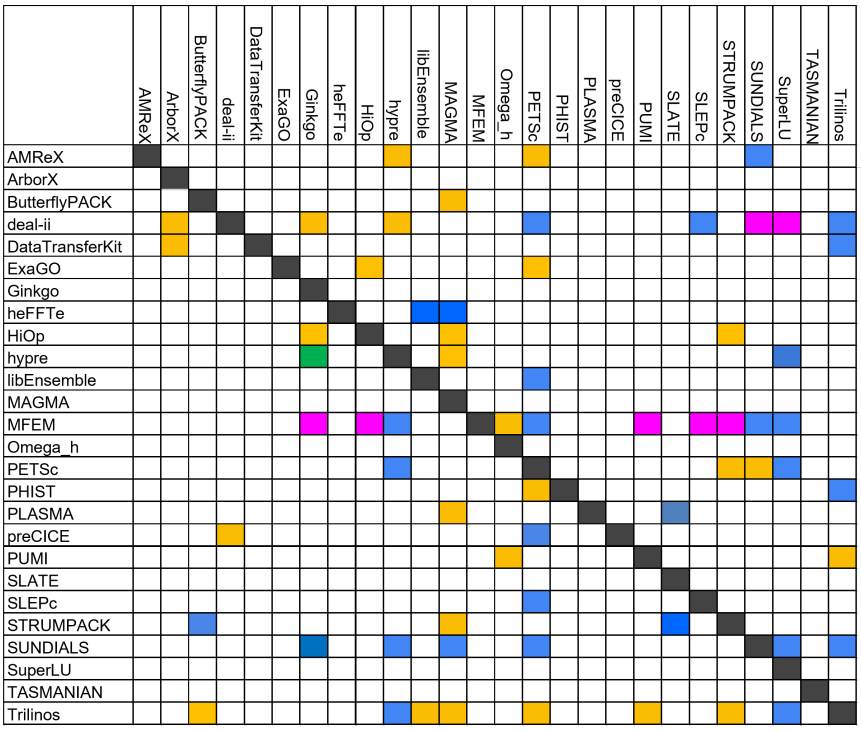

Fig. 3: Interoperability matrix for mathematical libraries in xSDK 1.0.0 (Nov. 2023). A filled in cell indicate that the row package can use capabilities from the column package e.g., AMReX can use linear solvers from *hypre* or PETSc or time integrators from SUNDIALS. Cell colors denote interoperability exists (yellow); interoperability exists and is enabled in the xSDK Spack package (blue); interoperability exists, is enabled in the Spack package, and is tested by the GitLab CI or in xsdk-examples (magenta); and interoperability is planned (green).

Today's complex multiphysics scientific and engineering simulation codes require a broad array of efficient implementations of numerical algorithms. As a result, many

of these codes combine multiple numerical methods from different libraries to help achieve their goals. Interoperability between these libraries, each targeting a particular need in the overall simulation code, has proven to be a significant challenge with problems falling in two main areas. First, packages must be able to be compiled together when generating a single executable. Issues often arise through namespace clashes, specific dependencies on software versions or compilers and their flags, or assumptions made at development time, like floating point precision. Many of the xSDK Community Policies were designed to address these challenges. The second challenge is in developing deeper interoperabilities between packages and largely stem from creating interfaces between packages to allow use of one package from another. Here decisions on how data is passed from one package to the other have significant bearing on the overall efficiency of the resulting capabilities, especially when the simulation code requires flexibility in the target computing platform architecture and programming model. Deeper interoperabilities, e.g., a common interface to several solver libraries, provide added functionalities for application codes and can accelerate application development.

The xSDK Community Policies work in conjunction with the Spack package manager to address the challenge of compiling the packages together. For example, policy M9 ensures namespace clashes do not happen, while policy R8 helps packages more easily support multiple versions of dependencies. With Spack, a package describes their dependencies, variants (compile-time options), and conflicts between variants, with specific versions of dependencies, compilers, etc. When installing a package with Spack, a user will tell Spack the abstract specification (spec) that they want to build and install, and Spack then turns this abstract spec into a concrete spec with all conflicts (to the extent possible based on the package description) resolved. The xSDK itself is a Spack meta-package which includes the individual xSDK packages as dependencies. The dependencies of the xSDK are described with abstract specs that are known to work together but are not unnecessarily descriptive. A typical dependency spec in the meta-package includes the version or version range of the dependency to be used, the floating point precision(s), the ordinal/index size(s), and relevant variants (such as GPU support) which may be enabled or disabled.

As noted above, to be a member package of the xSDK, there must be a deeper interoperability with at least one other member package. These interoperabilities have been greatly expanded in recent years to build greater flexibility and capability in the available numerical software stack. Figure 3 shows the current state of these deeper interoperabilities between the packages. A colored square in a row in this figure indicates that the package in the row name uses the package in the respective column name. In general, we see that discretization packages have fuller rows while solver packages have fuller columns. Colors indicate the maturity of the interoperability with magenta indicating that a deeper interoperability is in place, enabled in the xSDK Spack package, and regularly tested by the xSDK GitLab CI as discussed in Section 5. As the xSDK has matured, this matrix has become larger with more rows and columns and fuller with deeper interoperabilities in place.

A suite of example codes, called xsdk-examples,[37] demonstrate the use of various xSDK packages emphasizing the interoperability among packages to solve problems of interest. The examples provide both a training tool for users and a test suite as part of the testing of the xSDK that confirms correct xSDK builds. In its most recent release, xsdk-examples v0.4.0, the suite showcases 26 example codes, where 20 of the xSDK member packages are featured. Additionally, 10 examples are CUDA-enabled and 6 examples are HIP-enabled to demonstrate GPU interoperability between packages.

5 Release process and testing

Once a year, xSDK members create a new xSDK release that includes synchronized versions of all member packages and is installable with Spack. The most recent release, v1.0.0 from November 2023, can be installed via *spack install xsdk@1.0.0*. If one wants to build xSDK for NVIDIA, AMD, or Intel GPU architectures, additional specifications are required, as shown in the examples below:

spack install xsdk@1.0.0 +cuda cuda_arch=70
spack install xsdk@1.0.0 +rocm amdgpu_target=gfx90a
spack install xsdk@1.0.0 +sycl ~trilinos ~dealii %oneapi@2023.2.0

As the xSDK member packages are independent projects with developers working towards the packages' primary goals and requirements, the generation of a new release requires increased coordination between the individual packages. Participation within xSDK requires a public recent release of the package to be included in the new xSDK release. As part of this release process, installation and interoperability are tested to make sure the xSDK, as a whole, is installable via Spack. This process includes portability testing across multiple compilers (e.g., GNU, Clang, and Intel), workstations (e.g., Intel, AMD, and ARM CPUs), GPU platforms (i.e., NVIDIA/CUDA, AMD/ROCm and Intel/SYCL), and leadership computing facility (LCF) machines at ALCF, OLCF, NERSC, and LLNL. Any problems that come up need to be solved collectively by the xSDK project members. Each issue might need to be addressed and resolved by a different set of developers, thus improving the xSDK package ecosystem as a whole.

An early stage of xSDK release testing involves building development versions of packages against each other during their development cycles to identify regressions, API changes, or other updates that could potentially break interoperability between packages or the xSDK build as a whole. These jobs includes builds of different subsets of the xSDK that have dependencies between them. Detecting and fixing any problems that arise early in the release cycle can alleviate problems that would otherwise come up closer to the release date as packages coordinate on release versions. This testing is automated via GitLab CI which schedules and runs build tests daily.

[37] `https://github.com/xsdk-project/xsdk-examples`

As the xSDK release approaches, testing turns towards the full xSDK build and stabilization of individual package releases. This process is intertwined with the release process of individual packages with the goal of building the full xSDK collection to identify and address issues that need to be fixed before the release of individual member packages. This process includes changes to xSDK member package Spack recipes to enable building pre-release snapshots using the xSDK build process and is initially done with manual build testing on workstations. Once the builds can be automated, the tests are added to the GitLab CI. This automated testing captures new changes introduced in the member packages, the xSDK Spack meta package, or within Spack itself. This process is spread over several weeks with many xSDK member projects producing public releases at their convenient schedule, during this stage.

As the xSDK collection of package builds improves on workstations, it is also manually tested on LCF machines to find and address issues specific to those environments. LCF machines have more complex compiler and build software infrastructure compared to workstations as well as scheduler requirements, allocation limits, and wait times to access compute nodes. These extra considerations add to the complexity of testing xSDK builds on LCF systems. Moreover, xSDK builds are resource intensive. As some systems impose resource limits on the common use front-end nodes, xSDK builds are attempted on the corresponding compute nodes of such systems. Utilizing the compute nodes has the benefit of addressing potential differences in builds between compute and front end nodes. As member package developers might not have access to these resources, reproducing issues that come up during testing and addressing them further adds to the testing complexity.

After successful builds on workstations and LCF machines, the new xSDK version is released and made available via Spack, with documentation at xsdk.info/installing-the-software. Once the new version is available via Spack, additional testing of this release is scheduled via GitLab CI in order to test it against the daily changes that are added to Spack. Again any issues that come up are addressed and resolved.

6 Impact on applications

A variety of application codes—including ECP teams working in chemistry, materials, energy, Earth and space science, data analytics, optimization, AI, and national security [1] as well as many others in the broader community—have been and are benefiting from the greater availability of cutting edge numerical mathematical methods hardened into robust implementations through the establishment of the xSDK; whether it is from easier, more user-friendly builds of the whole application code or the availability of additional mathematical features from new library interoperability.

The first application to benefit from the xSDK was the one that sparked the whole endeavor, the Advanced Terrestrial Simulator or ATS[38] [19, 23], pictured in Figure

[38] `https://github.aamanzi.io`

1. It used the mathematical libraries *hypre*, Trilinos, PETSc, and SuperLU, and the domain components PFLOTRAN and Alquimia. The geochemistry reaction engine in PFLOTRAN was accessed through Alquimia. Updating the involved packages to adhere to the xSDK community policies removed version incompatibilities, namespace collisions, and other inter-package conflicts. The efforts also led to improved interoperability between some of the libraries and to the development of needed new features. With these improvements to the ecosystem, the application team was able to test a conceptual model of the Copper Creek catchment in the East River watershed, Colorado, in a quantitative manner with integrated-hydrology and a fully coupled surface/subsurface reactive transport model, see top left image in Figure 4. In this model, they simulated the hydrology, solute transport, and chemical reactive processes of 17 major chemical components in pyrite oxidation and calcite dissolution reactions along with more than 40 secondary species.

Another project that was able to take advantage of the xSDK is the ECP ExaWind team. This projects examines the flow physics governing whole wind plant performance, including wake formation, complex terrain impacts, and turbine-turbine interaction effects, see the top right image in Figure 4. The ExaWind team utilizes xSDK libraries AMReX, *hypre*, and Trilinos. The interoperability of xSDK libraries of Trilinos and *hypre* enabled the team to discover that *hypre* was able to solve some of their problems more efficiently.

The PeleLMeX[39] code [9] for simulating low-mach reacting flows, see the bottom image in Figure 4, also benefits from the xSDK ecosystem. PeleLMeX is built on top of the AMReX software framework for block-structured adaptive mesh refinement (AMR) applications and GPU portability. It utilizes AMReX's interoperability with *hypre* to solve elliptic systems that arise in a projection method to update advection velocities and with SUNDIALS for evolving numerous chemical reaction systems in time. When implicit time integration methods in SUNDIALS are employed to handle stiff chemistry mechanisms, PeleLMeX also utilizes SUNDIALS' interoperability with MAGMA and Ginkgo for solving linear systems. With the help from these libraries in the xSDK, PeleLMeX has demonstrated good scaling on the Frontier exascale supercomputer while simulating combustion processes typical of modern combustion systems [5].

In addition, xSDK played a critical role in the CEED[40] ECP co-design center, where *hypre* was vital in preconditioning matrix-free high-order finite element discretizations in NekRS and MFEM-based applications, including ExaSMR [41]. In addition to MFEM, the CEED software and applications also utilized GPU dense linear algebra routines from MAGMA, meshing algorithms from PUMI, linear algebra from PETSc and additional solvers from Gingko and STRUMPACK. See [13, 20, 2] for more details.

[39] https://github.com/AMReX-Combustion/PeleLMeX

[40] https://ceed.exascaleproject.org

[41] https://www.exascaleproject.org/research-project/exasmr/

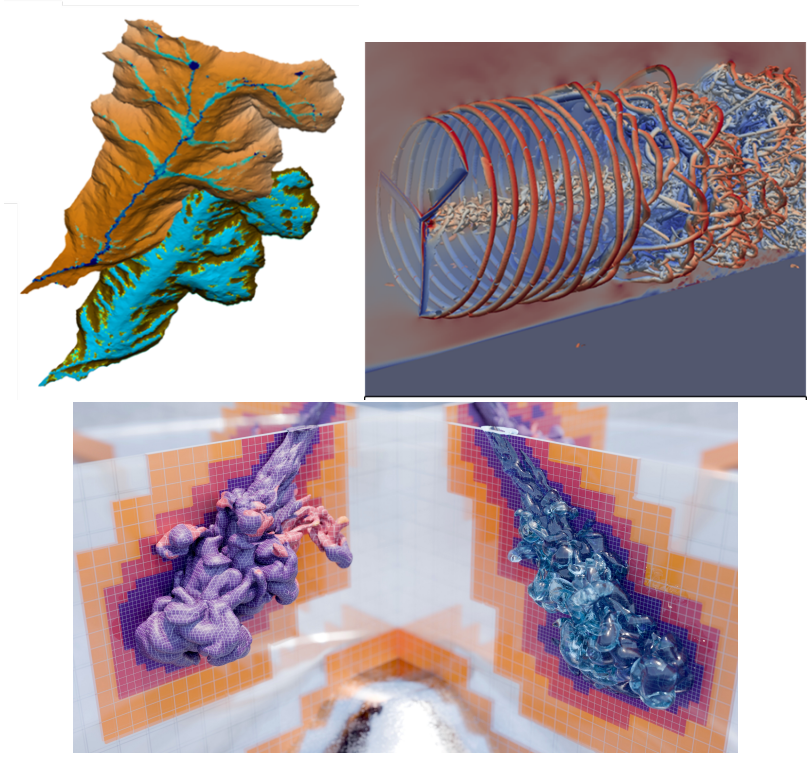

Fig. 4: Top left: Integrated hydrology simulation of Copper Creek shows the ponded depth of water on the surface (visualized above the subsurface), and the water saturation in the subsurface (Credit: David Moulton and Zexuan Xu). Top right: Flowfield, isosurfaces of Q-criterion colored by votricity magnitude and a plane with vorticity-magnitude iso-contours, for the NREL 5-MW rotor with rigid blades operating in uniform inflow of 8 m/s (Credit: Shreyas Ananthan, and Ganesh Vijayakumar). Bottom: Turbulent combustion of n-dodecane in a high-pressure piston chamber computed using PeleLMeX with an adaptive mesh hierarchy on Frontier (Credit: Nicholas Brunhart-Lupo and Marc Day).

7 Automatic performance (co-)tuning

In the xSDK ecosystem, apart from installation, testing and interoperability, another critical element is performance optimization—such as runtime, memory usage, energy usage—on an actual HPC machine. The performance of each library is a complex function depending on the input problems and the underlying computer architectures, including the inter-node communication network. Each library usually exposes a set of parameters to the users that can influence the library's performance. Most of the time the users simply apply default parameter settings

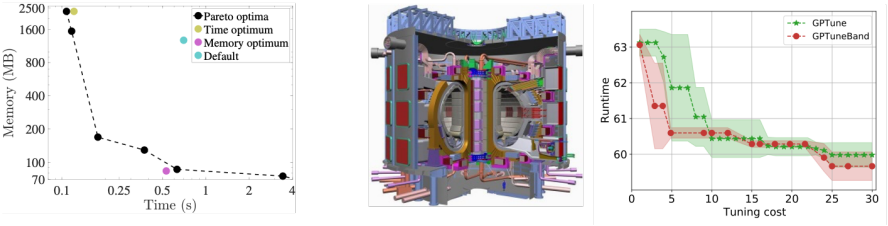

Fig. 5: GPTune tunning results: (left) two-objective tuning for SuperLU_DIST; (middle) ITER fusion reactor structure; (right) two algorithms in GPTune used to tune NIMROD for ITER two-fluid flow simulation.

which may be far from optimum. Worse yet, a bad choice of parameters may lead to non-convergence with solver libraries. When an application code uses multiple libraries, the dimension of the parameter space becomes intractably large for manually searches for the best configuration. We have been developing an autotuning tool to ease performance optimization tasks for users. Given the large number of libraries in the xSDK collection that provide diverse functionalities, the best strategy is to treat the performance optimization as "black-box" optimization. In particular, we employ the statistical Bayesian optimization framework to build performance surrogate models via Gaussian Process (GP) regression. The surrogate models are based on the execution results of the program collected on the actual machines with an automatically-chosen subset of parameter configurations. The tool is called GPTune. It features multitask/transfer-learning algorithms [15], multi-objective optimization, multi-fidelity tuning [25], clustered GP for non-smooth functions, hybrid models for mixture of real, integer and categorical variables [16], and a history database for crowd tuning.

We have applied GPTune to optimize parameters for the math libraries in xSDK as well as real simulation codes. Figure 5 shows the tuning results of two codes. The left subfigure shows pareto optimization for SuperLU_DIST with four parameters and two objectives: time and memory, for a sparse matrix Si2 from the PARSEC quantum chemistry application [15], using eight Cori nodes at NERSC with 256 MPI tasks. The default parameter setting is about 5× slower and uses 14× more memory compared to the tuned setting. The right subfigure shows the tuning of the multiphysics simulation code NIMROD to predict microscopic MHD instabilities of burning plasma in the ITER fusion reactor (middle). The code uses SuperLU_DIST as a preconditioner for the GMRES iterative linear solver. We co-tuned five parameters, 2 from NIMROD and 3 from SuperLU_DIST, and achieved an approximately 10% performance gain on 16 Cory nodes with 512 MPI tasks. The plot also shows that the multi-fidelity algorithm GPTuneBand is superior to the vanilla single-fidelity GPTune [25]. Another demonstration of GPTune can be seen in the paper on the application of Anderson acceleration for fusion applications in this volume [10].

8 Conclusion

This article discusses the motivation for and many benefits of an ecosystem of high-performance numerical mathematics libraries for scientific applications, as achieved through the xSDK. Development of the xSDK has been a critical step in getting new, highly performant, and robust implementations of mathematical methods into the hands of applications scientists to use in their simulations. These implementations are the products of decades of numerical methods research. However, as noted throughout this paper, there are significant challenges in ensuring these implementations work well together. The xSDK and the work discussed in this paper on policies, interoperabilities, testing, releases, deploying to applications, and performance tuning are critical to ensuring new numerical methods are highly performant on today's computing platforms, performing as an ecosystem of numerical libraries. With the conclusion of the DOE Exascale Computing Project in December 2023, work on the stewardship of numerical libraries and other ECP software technologies is transitioning to the newly established Consortium for the Advancement of Scientific Software (CASS) [7]. To address the continuing challenges of heterogeneous computer architectures at all scales of computing and the increasing complexities of application codes, we see a growing need for even broader collaboration on numerical software ecosystems [4] throughout the international community.

Acknowledgements This work has been 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. This work was performed in part under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC,(LLNL-JRNL-864981), by Lawrence Berkeley National Laboratory under contract No. DE-AC02-05CH11231.

Research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as presenting the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

Support for this work was provided in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program through the Frameworks, Algorithms, and Scalable Technologies for Mathematics (FASTMath) Institute.

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of EnergyâĂŹs National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is co-authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.

We would like to thank the users of xSDK libraries for motivating advances in library functionality and the developers of all the xSDK libraries for engaging in collaboration on community policies and other software advances. We also thank the reviewers for their helpful comments on how to improve the paper.

Competing Interests The authors have no conflicts of interest to declare that are relevant to the content of this chapter.

References

1. Alexander, F., et al.: Exascale applications: skin in the game. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **378**(2166), 20190056 (2020). DOI 10.1098/rsta.2019.0056
2. Andrej, J., Atallah, N., Dobrev, V., Kolev, T., Pazner, W., Tomov, V., et al.: High-performance finite elements with MFEM. The International Journal of High Performance Computing Applications (2024). To appear
3. Anzt, H., Huebl, A., Li, X.S.: Then and now: Improving software portability, productivity, and 100× performance. IEEE Comput. Sci. Eng. **25**(6) (2023). DOI 10.1109/MCSE.2024.3387302.
4. Bader, M., Ltaief, H., McInnes, L., Yokota, R.: How we can all play the high-performance game: From laptops to supercomputers. SIAM News (2023)
5. Balos, C.J., Day, M., Esclapez, L., Felden, A.M., Gardner, D.J., Hassanaly, M., Reynolds, D.R., Rood, J., Sexton, J.M., Wimer, N.T., Woodward, C.S.: SUNDIALS time integrators for exascale applications with many independent ODE systems. arXiv preprint arXiv:2405.01713 (2024)
6. Bartlett, R., Demeshko, I., Gamblin, T., Heroux, M., Johnson, J., Klinvex, A., Li, X., McInnes, L., Moulton, J., Osei-Kuffuor, D., Sarich, J., Smith, B., Willenbring, J., Yang, U.: xSDK Foundations: Towrd an Extreme-scale Scientific Development Kit. Supercomputing Frontiers and Innovation **4**, 69–82 (2017)
7. Consortium for the Advancement of Scientific Software (CASS). `https://cass.community`
8. DOE Exascale Computing Project (ECP). `https://www.exascaleproject.org`
9. Esclapez, L., Day, M., Bell, J., Felden, A., Gilet, C., Grout, R., de Frahan, M.H., Motheau, E., Nonaka, A., Owen, L., Perry, B., Rood, J., Wimer, N., Zhang, W.: PeleLMeX: An AMR low mach number reactive flow simulation code without level sub-cycling. Journal of Open Source Software **8**(90), 5450 (2023). DOI 10.21105/joss.05450
10. Gardner, D.J., LoDestro, L.L., Woodward, C.S.: Towards the use of Anderson acceleration in fusion simulations. In: This volume (2024)
11. Henderson, M.E., Anderson, C.R., Lyons, S.L., et al.: Object Oriented Methods for Interoperable Scientific and Engineering Computing: Proceedings of the 1998 SIAM Workshop, vol. 99. SIAM (1999)
12. Heroux, M., Willenbring, J., Shende, S., Coti, C., Spear, W., Peyralans, J., Skutnik, J., Keever, E.: E4S: Extreme-Scale Scientific Software Stack. In: 2020 Collegeville Workshop on Scientific Software Whitepapers (2020)
13. Kolev, T., Fischer, P., Min, M., Dongarra, J., Brown, J., Dobrev, V., Warburton, T., Tomov, S., Shephard, M.S., et al.: Efficient exascale discretizations: high-order finite element methods. The International Journal of High Performance Computing Applications pp. 527–552 (2021). DOI 10.1177/10943420211020803
14. Kothe, D., Lee, S., Qualters, I.: Exascale computing in the United States. Computing in Science and Engineering **21**(1), 17 – 29 (2019). `https://doi.org/10.1109/MCSE.2018.2875366`
15. Liu, Y., Sid-Lakhdar, W., Marques, O., Zhu, X., Meng, C., Demmel, J., Li, X.: GPTune: multitask learning for autotuning exascale applications. In: PPoPP '21: Proceedings of the

26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 234–246. San Francisco, USA (2021). DOI 10.1145/3437801.3441621
16. Luo, H., Cho, Y., Demmel, J.W., Li, X.S., Liu, Y.: Hybrid parameter search and dynamic model selection for mixed-variable Bayesian optimization. Computational and Graphical Statistics (2024). DOI 10.1080/10618600.2024.2308216
17. McInnes, L.C., Heroux, M.A., Bernholdt, D.E., Dubey, A., Gonsiorowski, E., Gupta, R., Marques, O., Moulton, J.D., Nam, H.A., Norris, B., Raybourn, E., Willenbring et al., J.: A cast of thousands: How the IDEAS productivity project has advanced software productivity and sustainability. IEEE Comput. Sci. Eng. **25**(6) (2023). DOI 10.1109/MCSE.2024.3383799
18. McInnes, L.C., Heroux, M.A., Draeger, E.W., Siegel, A., Coghlan, S., Antypas, K.: How community software ecosystems can unlock the potential of exascale computing. Nature Computational Science **1**, 92–94 (2021). `https://doi.org/10.1038/s43588-021-00033-y`
19. Molins, S., Svyatsky, D., Xu, Z., Coon, E., Moulton, J.: A multicomponent reactive transport model for integrated surface-subsurface hydrology problems. Water Resources Research **58** (2022). DOI 10.1029/2022WR032074
20. Vargas, A., Stitt, T., Weiss, K., Tomov, V., Camier, J.S., Kolev, T., Rieben, R.: Matrix-free approaches for GPU acceleration of a high-order finite element hydrodynamics application using MFEM, Umpire, and RAJA. The International Journal of High Performance Computing Applications **36**(4), 492–509 (2022)
21. Willenbring, J.M., Shende, S.S., Gamblin, T.: Providing a flexible and comprehensive software stack via Spack, E4S, and SDKs. IEEE Comput. Sci. Eng. **25**(6) (2023). DOI 10.1109/MCSE.2024.3395016
22. xSDK: Extreme-scale Scientific Software Development Kit. URL `https://xsdk.info/`
23. Xu, Z., Molins, S., nI. Dwivedi, Svyatsky, D., Moulton, J., Steefel, C.: Understanding the hydrogeochemical response of a mountainous watershed using integrated surface-subsurface flow and reactive transport modeling. Water Resources Research **58** (2022). DOI 10.1029/2022WR032075
24. Yang, U., McInnes, L.: xSDK: Building an ecosystem of highly efficient math libraries for exascale. SIAM News **54**, 8–10 (2021)
25. Zhu, X., Liu, Y., Ghysels, P., Bindel, D., Li, X.: GPTuneBand: multi-task and multi-fidelity autotuning for large-scale high performance computing applications. In: SIAM PP22: Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing (2022)