The PETSc Community as Infrastructure

Mark Adams, *Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA*

Satish Balay, Oana Marin, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Hong Zhang, and Junchao Zhang, *Argonne National Laboratory, Lemont, IL, 60439, USA*

Jed Brown, *University of Colorado at Boulder, Boulder, CO, 80309, USA*

Victor Eijkhout, *The University of Texas at Austin, Austin, TX, 78712, USA*

Jacob Faibussowitsch, *University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA*

Matthew Knepley, *University of New York at Buffalo, Buffalo, NY, 14260, USA*

Fande Kong, *Idaho National Laboratory, Idaho Falls, ID, 83415, USA*

Scott Kruger, *Tech-X Corporation, Boulder, CO, 80303, USA*

Patrick Sanan, *ETH Zurich, 8092 Zürich, Switzerland*

Barry F. Smith, *Flatiron Institute, New York, NY, 10010, USA*

Hong Zhang, *Illinois Institute of Technology, Chicago, IL, 60616, USA*

The communities that develop and support open-source scientific software packages are crucial to the utility and success of such packages. Moreover, they form an important part of the human infrastructure that enables scientific progress. This article discusses aspects of the Portable Extensible Toolkit for Scientific Computation community, its organization, and technical approaches that enable community members to help each other efficiently and effectively.

To meet the technological challenges of the 21st century, the simultaneous revolutions in data science and computing architectures need to be mirrored by a revolution in scientific simulation that provides flexible, scalable, multiphysics multiscale capabilities in both traditional and new areas. This simulation technology rests on a foundation of numerical algorithms and software for high-performance computing. This foundation raises in importance to the level of classical hard infrastructures but it requires human investment and new ways of organizing the effort for software and algorithm development, support, and maintenance.

Much simulation technology today is developed and supported with a *community*[a] *open-source* software paradigm.[1,3,5,9] Many numerically oriented open-source projects, including SciPy, Julia, and Stan, thrive because of their communities; those without a community die out, have only a fringe usership, or are maintained (as orphan software) by other communities.

In addition, an explicit focus on software ecosystems—collections of interdependent products whose development teams have incentives to collaborate to provide aggregate value—is addressing growing HPC complexity.[6] Notable efforts include the xSDK, where community policies[b] are helping with coordination among numerical packages, and E4S,[c] a broader effort addressing functionality across the HPC software stack. These endeavors are funded by the U.S. Department of Energy (DOE) within the Exascale Computing Project (ECP).[d] Moreover, communities are tackling challenges in *how* research software is developed and sustained[10] as well as software stewardship.[8]

[a]The source is developed in a public environment with contributors from a variety of institutions.

[b]https://xsdk.info/policies
[c]https://E4S.io
[d]https://exascaleproject.org

Digital Object Identifier 10.1109/MCSE.2022.3169974
Date of publication 11 May 2022; date of current version 31 August 2022.

This article presents an explanatory case study of the Portable, Extensible Toolkit for Scientific Computation (PETSc),[2] considering *community as infrastructure*. PETSc began in the early 1990s at Argonne National Laboratory as a project for research on parallel numerical algorithms. Since then, developers, users, and functionality have grown substantially, driven by continually expanding community needs to exploit advances in HPC architectures for next-generation science fully. The authors have over 160 years of combined experience with PETSc; their training ranges from mathematics to computer science to science and engineering. One author also supports PETSc at a supercomputing center; another maintains the testing and merge request infrastructure. Several authors are liaisons with other software communities. PETSc comprises software infrastructure (code and tools) plus human infrastructure: the community of people who develop, support, maintain, use, and fund PETSc, their interactions, and their culture. The human infrastructure—people and their interactions as a community, within the broader DOE, HPC, and computational science communities—is foundational and enables the creation of sustainable software infrastructure.

PETSc was not originally purposefully designed to support long-term community software infrastructure. Rather, work on the software inspired the creation of a set of practices to enable a small development team with large ambitions and a long time horizon to develop and support software capable of solving problems of interest to the developers and their collaborators. However, these practices, reviewed in the following, have wider benefits, and certain community properties could serve as a template for long-term software infrastructure:

- enabling swift, in-depth engagement, especially for new users;
- encouraging and offering opportunities for anyone to contribute to the software and documentation;
- providing a virtual institution for collaboration;
- developing extensible interfaces that enable people interested in mathematics and algorithms to experiment and deploy research;
- supporting developer autonomy to pursue topics aligned with individual research needs;
- enabling strong ties to academia, industry, and laboratories worldwide;
- committing to continually advancing library capabilities as needed by next-generation science and HPC architectures.

Spread throughout the world, the PETSc community allows the real-time transfer of knowledge across institutions and application fields. Also, community interactions promote algorithmic development, enable state-of-the-art advances, and benefit the scientific community.

We organize this article as follows. In the next section, we discuss the purposes of the PETSc community and the various roles that members play. After that, we introduce several key organizational principles and communication patterns. We then introduce the paradigm of *debugging by e-mail*, which encapsulates the philosophy and software technologies we use to help each other (regardless of location) use, debug, and improve PETSc. This section shows how technical choices in design and software details can be made specifically to enhance the community experience.

COMMUNITY

This section outlines the myriad purposes of the PETSc software and its roles within the PETSc community. First, of course, the PETSc community (similar to other package communities) is embedded in the DOE, HPC, and broader computational science communities.

Purposes of PETSc

PETSc serves many purposes as a software library that connects research in applied mathematics to usage within applications in science and engineering. These include:

- a *research platform* targeting innovative algorithmic development;
- a well-supported *HPC library*;
- a *repository of template applications* via a wealth of example codes;
- a *compendium of algorithms*, with an algorithmic management system that provides concrete, scalable implementations of a wide range of methods described in the applied mathematics literature;
- an *application development framework*;
- a *pedagogical tool* for training numerical analysts on HPC platforms[4];
- a *source of best choice numerical methods* in its role as an interface between academic algorithmic development and the needs of users in science and engineering;
- an extensible *interface to complementary HPC software*, such as SuperLU and hypre.

Roles of PETSc Community Members

Virtually all active PETSc community members are PETSc users; a smaller subset of these, who often

began as PETSc users, are also PETSc developers. All PETSc users provide important contributions, including bug reports, bug fixes, improved documentation, and suggestions for new features. Individuals often move between different roles in the PETSc community. There is a "long tail" who contributes less frequently than the most active developers, yet collectively contributes a great deal. Over one hundred people are contributors to the PETSc Git repository; hundreds more communicate through e-mail and GitLab issues each year, and thousands use PETSc directly or via other toolkits.

The community structure is crucial in providing a pathway to increasing involvement for interested users. One way is to characterize PETSc community members along institutional lines.

- *Academic-oriented users* are students, faculty, and staff focusing on research and development, employed by universities and research laboratories. Students may use PETSc to do homework or develop a paper or thesis code. Students often contribute code back to PETSc, and, as they graduate, bring PETSc to their new institutions. Some students have become PETSc developers.
- *Industrial users* employ PETSc in their company's research or commercial products. PETSc's 2-clause BSD license eases its commercial use. These users may request support that is unlikely to be funded by research grants, such as support for Microsoft Windows; fortunately, there are avenues for PETSc community members to help with these requests. Industrial users require discretion and confidentiality. They cannot always share their use cases, so we must provide general solutions without details of the specifics. Developing the trust needed for industrial users is a gradual process whose importance must be recognized by both sides.

Another way to categorize PETSc community members is by the goals of their work, as shown in Figure 1.

- *Algorithm developers* focus on devising and analyzing algorithms and hence may be less concerned about generality and usability. They use PETSc because it provides infrastructure for HPC architectures, allowing them to avoid unnecessary coding. Algorithm developers face the challenge of writing scalable implementations, which, even in PETSc, can be time consuming with a steep learning curve. However, such people find the benefits of using PETSc, including the broad impact of their work on HPC applications, outweigh using packages with a less steep learning curve, such as MATLAB.
- *Scientific toolkit developers* build systems that tackle a subset of PETSc functionality but at a higher level of abstraction, with more specific support for their target class of problems. Such toolkits, including Firedrake, MOOSE, and Deal.II, leverage PETSc capabilities and introduce additional infrastructure.
- *Application developers* focus on creating a code that addresses one specific simulation. They are often discipline scientists or engineers, who benefit from performance enhancements provided through PETSc composability, where upgrades in algorithms and data structures can occur seamlessly from a user perspective, yet provide significant performance increases.

ORGANIZATION AND COMMUNICATION

We now summarize the organizational and communication patterns in PETSc due to its various purposes and member roles. Besides communicating within funded projects, institutional settings, and events, users engage in the PETSc community through online support mailings, GitLab issues, and Slack channels. Annual PETSc user meetings include tutorials on leveraging library functionality for research while also highlighting users' science achievements made possible by advances in PETSc features.

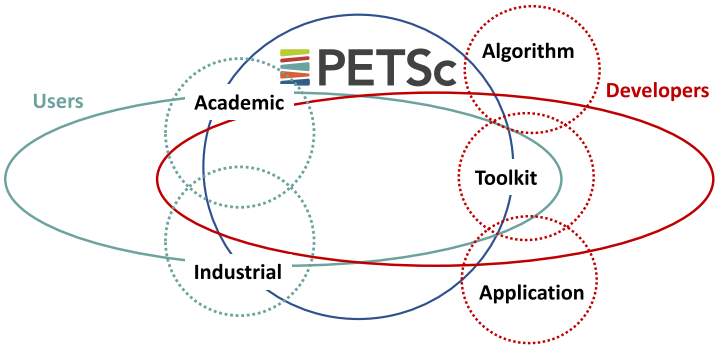

FIGURE 1. PETSc is a scientific junction, where the boundary between types of users and developers is fluid; roles shift and change.

Engagement When Problems Occur

Support is a crucial aspect of a healthy software community. Members of the PETSc community usually respond to requests within hours, if not minutes. This engagement helps new users feel welcomed and valued, make rapid progress, and gain confidence. Through users' feedback, developers learn what works, what does not, and where improvements are needed. PETSc community members discover new research topics from feature requests and discussions.

However, providing excellent support is a substantial effort, particularly when users encounter difficult bugs or performance issues at scale. User-developer communication on a particular topic can span weeks or even months. PETSc developers need to be patient and consistently engaged with users. One may question whether this practice is sustainable, but it has worked reasonably well. In the last section, we discuss some technical approaches to providing support.

Trust Within the Community

To maintain the vitality of the library, new algorithmic developments must be rapidly integrated, bugs promptly fixed, and awkward constructions removed. These activities require the PETSc community to establish a high level of trust, communicating that the library will be well supported even in the face of rapid evolution, and that code will continue to run with help from the community. Members of the PETSc community have a wide range of professions, backgrounds, and levels of involvement, with individuals often participating in several ways over the years. Engagement is key to disseminating tacit knowledge and developing users' skills and social support so that people can transition to become developers and mentors. The PETSc community has developed a broad base of people with expertise and kindness to reduce and report bugs, mentor newcomers, and contribute in other ways. To help improve the atmosphere, which can be difficult for newcomers, the PETSc community has adopted a code of conduct.[e]

Community to Community

Application communities often treat PETSc as a software ecosystem instead of a stand-alone package. As a result, they rely on PETSc to manage necessary low-level tools, such as MPI, BLAS/LAPACK, and vendor packages used on accelerators. Application communities also appreciate unified solver interfaces, particularly linear preconditioners, which enable application codes to access third-party libraries, such as MUMPS, SuperLU, and hypre with little effort. Often "technical language" barriers exist between communities. For example, an expert in contact mechanics may describe a solver convergence issue as "we found a PETSc error when contacts occur with a frictionless model." A solvers expert might find it hard to resolve such an issue. Fortunately, other PETSc community members may have domain expertise and can serve as liaisons between communities. Such individuals speak the languages of both communities, understanding both PETSc's capabilities and the needs of their communities. Thus, they can explore, explain, and introduce PETSc features to their communities. Such liaisons help expand PETSc's reach across disciplines and reduce the centralized maintenance burden by addressing many questions directly in their communities while contributing patches, feature requirements, and even serving as testers of software releases.

An important PETSc subcommunity is systems engineers who manage institutions' computational infrastructure. They often are the first to encounter problems that need the attention of PETSc developers. Their expertise can help rapidly debug problems and develop fixes. Package maintainers, for example, for APT, are also a valuable resource, as they track PETSc on particular configurations; they often have excellent suggestions for improvements to PETSc's configuration and installation.

Responding to Change

Communities must respond with innovative and creative solutions to changing circumstances. For numerical software, this includes the continual emergence of new science drivers and techniques, currently data science and artificial intelligence, as well as new hardware architectures. For example, a large shift in HPC is underway with incorporating graphical processing units (GPUs) into scientific computing. Major organizations, such as DOE have responded with, for example, the ECP, where community open-source projects, including PETSc, are aggressively developing innovations in data structures and algorithms for new architectures,[7,11] The PETSc community empowers developers to be creative by providing the autonomy to be innovative while still maintaining guidelines for development[f] and town squares to organize overall development plans. This approach, along with PETSc's wide variety of contributors, enables a level of agility that might not otherwise occur. This approach also promotes project-specific planning (for example, as needed for work proposed and funded in particular

[e]https://gitlab.com/petsc/petsc/-/blob/main/CODE_OF_CONDUCT.md

[f]https://petsc.org/release/developers

grants) and coordination among development communities overall.

Enabling Research Collaborations

PETSc's community helps members identify funding opportunities, access expertise, and transition between roles in the project. One of the greatest difficulties in maintaining a coherent software project over decades is providing career paths for contributors. PETSc gives academic contributors a solid foundation for advancement via awards (e.g., SC Gordon Bell prizes and SIAM prizes), the highly cited users' manual, professional recognition, and productive collaborations born from PETSc development, maintenance, and support. In addition, PETSc provides resource sharing from collaborative grants and collaboration opportunities that extend beyond the development group. Sometimes, PETSc affiliation may be more important than departmental affiliation, especially since modern academic departments are often atomized, with little internal collaboration. The community provides strong academic connections for industrial and laboratory members, tangible outputs recognized by future employers, and active participation in the wider computational science community.

Engagement With Funding Institutions

Even the smallest community open-source projects cannot exist without some funding and institutional support. Usually, it is a combination of grants from governmental or nongovernmental agencies, in-house funding within particular institutions, and less formal systems that allow employees to contribute to open-source packages during a portion of their regular employment. Members of the PETSc community are actively engaged with program development, including communicating with program managers at the U.S. Department of Energy, the National Science Foundation, and with institutional management to ensure that support is provided and maintained. This form of interaction is crucial to the long-term viability of all open-source software communities; PETSc users and community members have played important roles in various local, national, and international conversations, including recent DOE activities related to software sustainability.[8,10]

SOFTWARE DESIGN TO SUPPORT THE COMMUNITY

Helping people when they encounter problems is essential for the PETSc community. Members of the community need to diagnose quickly where problems occur and refer users to corresponding support if a solution is beyond the expertise of PETSc developers. Because of its complexity—users call PETSc through its Python, Fortran, or C bindings; in Linux, macOS, or Windows operating systems; on machines from laptops to the world's most powerful supercomputers; on x86, Arm, Power, or other CPU architectures, possibly accelerated by GPUs from different vendors—PETSc community helpers cannot reproduce all problems met by individuals. Thus, PETSc has a holistic remote debuggable design, enabling a feature dubbed *debugging by e-mail*, so PETSc community members can pinpoint causes of problems through conversations in PETSc mailing lists and GitLab issues.

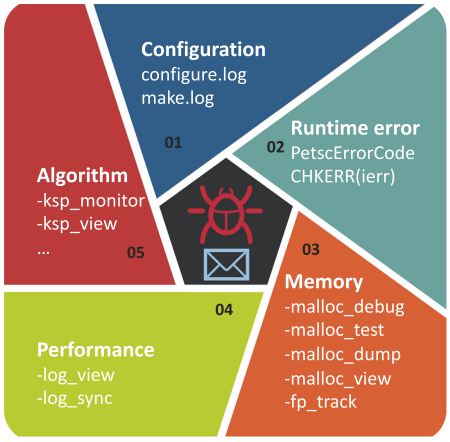

FIGURE 2. PETSc has a holistic remote debuggable design, enabling a feature dubbed *debugging by email*, so PETSc community members can pinpoint causes of problems through conversations in PETSc mailing lists and GitLab issues.

This approach enables the community to do more with less. In this section, we introduce the debuggability design of PETSc, which is designed and implemented to improve engagement and support. PETSc's debuggability spans from configuration to code execution, with the salient features highlighted in Figure 2. Our commitment to software support leads us to maintain our own simple, integrated tools for many tasks that conventional software wisdom would dictate should be performed exclusively by full-featured external tools. The combination of internally developed and external tools used in PETSc is unique to its history and "high-end" HPC focus and is not necessarily the best approach for other open-source packages.

Configuration Debugging

HPC software systems have complex execution environments, including great variance in hardware and

```
1  PetscErrorCode  VecAXPY(Vec y,PetscScalar alpha,Vec x){
2    PetscErrorCode ierr;
3    PetscFunctionBegin;
4    PetscValidHeaderSpecific(x,VEC_CLASSID,3);
5    PetscValidType(x,3);
6    ...
7    PetscCheckSameTypeAndComm(x,3,y,1);
8    VecCheckSameSize(x,3,y,1);
9    if (x == y) SETERRQ(comm,PETSC_ERR_ARG_IDN,"x and y cannot be the same vector");
10   PetscValidLogicalCollectiveScalar(y,alpha,2);
11   ...
12   ierr = PetscLogEventBegin(VEC_AXPY,x,y,0,0);CHKERRQ(ierr);
13   ierr = (*y->ops->axpy)(y,alpha,x);CHKERRQ(ierr);
14   ierr = PetscLogEventEnd(VEC_AXPY,x,y,0,0);CHKERRQ(ierr);
15   ...
16   PetscFunctionReturn(0);
```

LISTING 1. Sample showing error checking in PETSc.

software. Software developers must expend considerable effort to configure and build their code for different situations. When failures occur, the software developers need to have information available in a usable format to diagnose and fix the problems. Configuration failures are the most common support issues the PETSc community faces. Thus, having a debuggable configuration system with comprehensive logging is critical. Rather than using a standard configuration system, such as GNU Autotools or CMake, PETSc, has a bespoke configuration system with extensive checking, written in Python, which logs everything during configuration in a single file `configure.log`. When a check fails, it generates clear error messages and a Python stack trace. PETSc users attach `configure.log` and another file `make.log` generated by `make` when they meet configuration errors. By examining the two files, PETSc developers can quickly determine why the configuration system made specific choices and what went wrong. In addition, since the configuration system is bespoke, the PETSc community can easily add new checking, testing, and logging. CMake is notoriously difficult to debug by e-mail because it logs information in various directories and does not log much of its process.

Runtime Error Debugging

The PETSc library strives to provide descriptive error messages that explain why and where errors have occurred, making it easy for PETSc developers to diagnose by e-mail what went wrong and assist users with fixes. PETSc has extensive code to assist in this regard. See Listing 1, which shows code that adds two vectors with `y += alpha*x`. Every PETSc function returns a `PetscErrorCode`, indicating whether the function is successfully executed, and if not, what error occurred. In PETSc source code, every function call is error checked, as in lines 12–14. We make errors manifest early rather than later to avoid obscure error messages. The default error handler prints the stack trace leading to the error, including function names, file names, and line numbers. The stack trace is built inside the two macros `PetscFunctionBegin` and `PetscFunctionReturn(0)`; see lines 3 and 16. PETSc also provides utilities to check the integrity of function parameters; see lines 4–10. All application programming interfaces (APIs) shown here are public; users are encouraged to apply the same strategy in their code.

PETSc also has APIs to assert properties of the code so that useful error messages are generated promptly if the code behaves unexpectedly. For example, once a matrix is preallocated or assembled, one can set a property of the matrix to indicate that in subsequent insertions one will insert only to existing nonzero locations.

Memory Debugging

Memory corruption problems are common; therefore, memory allocations are done through a PETSc-specific API that records information and sets sentinels around the allocations in debug mode. With command-line options, PETSc will initialize the allocated memory with not a number; using the uninitialized memory in floating-point operations will generate an appropriate error message. Also, PETSc can check the integrity of the entire heap of PETSc-allocated memory at every allocation. PETSc codes also can output information about memory that has never been freed during the PETSc finalization stage, to detect memory leaks. As lightweight Valgrind-like features, the output can be shared with PETSc developers to help understand a code's misbehavior. But, of course, we also recommend using more sophisticated tools, including Valgrind and debuggers.

```
Summary:  ----- Time ------  ----- Flop ------  --- Messages ---  -- Message Lens  -- Reductions --
            Avg    %Total     Avg   %Total    Count   %Total     Avg    %Total    Count   %Total
  Setup:  2.398e-03  33.6%  4.608e+03   0.6%  1.200e+01   1.9%  9.300e+02  8.4%   2.300e+01   9.1%
  Solve: 4.729e-03  66.3%  8.183e+05  99.4%  6.200e+02  98.1%  1.963e+02 91.6%   2.100e+02  83.3%

Phase summary info:
   Count: number of times phase was executed
   Time and Flop: Max - maximum over all processors
                  Ratio - ratio of maximum to minimum overall processors
   Mess: number of messages sent
   AvgLen: average message length (bytes)
   Reduct: number of global reductions
   Global: entire computation
   Stage: stages of a computation. Set stages with PetscLogStagePush() and PetscLogStagePop().
      %T - percent time in this phase         %F - percent flop in this phase
      %M - percent messages in this phase     %L - percent message lengths in this phase
      %R - percent reductions in this phase
   Total Mflop/s: 10e-6 * (sum of flop over all processors)/(max time overall processors)
------------------------------------------------------------------------------------------------------
Event                Count      Time (sec)     Flop                              --- Global ---  Total
                   Max Ratio  Max     Ratio   Max  Ratio  Mess   AvgLen  Reduct  %T %F %M %L %R Mflop/s
------------------------------------------------------------------------------------------------------
MatMult               54 1.0 4.4847e-04 1.5 8.86e+04 1.1 3.4e+02 1.6e+02 1.0e+00  5 42 53 39  0   762
MatMultAdd             5 1.0 7.3937e-05 4.3 2.13e+03 1.1 1.5e+01 9.1e+01 0.0e+00  1  1  2  1  0   111
MatMultTranspose       5 1.0 4.6906e-05 2.2 2.23e+03 1.1 2.1e+01 7.2e+01 1.0e+00  0  1  3  1  0   179
VecAXPY               48 1.0 9.4440e-06 1.8 9.60e+03 1.0 0.0e+00 0.0e+00 0.0e+00  0  5  0  0  0 4066
VecScatterBegin       73 1.0 2.6231e-04 1.6 0.00e+00 0.0 4.5e+02 1.6e+02 5.0e+00  3  0 72 53  2     0
VecScatterEnd         73 1.0 4.4319e-04 3.6 1.70e+02 0.0 0.0e+00 0.0e+00 0.0e+00  4  0  0  0  0     0
KSPSetUp               5 1.0 6.2018e-04 2.8 4.53e+04 1.0 6.0e+01 1.6e+02 3.6e+01  5 22  9  7 14   288
KSPSolve               1 1.0 1.0645e-03 1.0 1.26e+05 1.0 2.4e+02 1.5e+02 1.4e+01 15 60 38 27  6   466
```

LISTING 2. Sample output of -log_view.

Performance Debugging

Another challenging support task, which the PETSc community also handles routinely, is debugging performance problems, particularly for high levels of parallelism. PETSc provides APIs to allow developers and users to set stages in their code and log the performance of events of interest. For example, lines 12 and 14 in Listing 1 are for the `VEC_AXPY` event, rendering a lightweight, integrated logging system that allows users to quickly gather timings. Listing 2 shows a snippet of the stdout output. From the top, we know that the computation has two stages, labeled as setup and solve. PETSc summarizes the computation and communication statistics of the two stages. Below that, it lists detailed statistics of events (functions) within each stage (only the first stage is shown), including the number of times an event has been called, time and floating-point operations (flops) an event has spent, MPI messages and reductions an event has incurred, and other statistics. Interested readers are referred to the PETSc/TAO Users Manual[2] or hints in the log view message itself. Because MPI processes have different statistics in parallel, PETSc shows maxima overall processes and ratios of maxima to minima. This information is useful because whenever we encounter a large ratio in time or flops in output, we know a load imbalance in the corresponding event might exist. Sometimes, imbalance in one event can distort the timing of other events (for instance, processes might wait for messages from a lagging partner), giving confusing results.

Having the profiling tools integrated with the numerical algorithms in use, outputting by default to `stdout`, is crucial because it allows all users to provide quickly information on their usage, independent of what computational systems they may use or which additional analysis or logging tools they have available. This allows PETSc developers to quickly and directly view timings on the user's system and facilitate performance debugging of scalable solvers at "production" scale, by e-mail, where direct reproduction of a user's issue is infeasible.

Algorithm Debugging

PETSc includes an extensive suite of parallel preconditioners, linear solvers, nonlinear solvers, and time integrators. Composable and nested solvers are among the most powerful PETSc features since they facilitate numerical experimentation on a novel, complex problems, but keeping track of them can be difficult. PETSc developers must see the detailed solver configurations to spot potential problems. Hence, PETSc provides APIs to display all solver options being used. Listing 3 shows a snippet of a longer output from a nonlinear solver. With indention reflecting levels of the composite solvers, we can see the nested solvers used and key parameters employed at various levels of the solvers.

```
SNES Object: 4 MPI processes
  type: newton linesearch
  maximum iterations=2, maximum function evaluations=10000
  tolerances: relative=1e-10, absolute=1e-50, solution=1e-08
  total number of linear solver iterations=14
  total number of function evaluations=2
  norm schedule ALWAYS
  SNESLineSearch Object: 4 MPI processes
    type: bt
      interpolation: cubic
      alpha=1.000000e-04
    maxstep=1.000000e+08, minlambda=1.000000e-12
    tolerances: relative=1.000000e-08, absolute=1.000000e-15, lambda=1.000000e-08
    maximum iterations=40
  KSP Object: 4 MPI processes
    type: cg
    maximum iterations=100, initial guess is zero
    tolerances:  relative=1e-10, absolute=1e-50, divergence=10000.
    left preconditioning
    using UNPRECONDITIONED norm type for convergence test
  PC Object: 4 MPI processes
    type: gamg
      type is MULTIPLICATIVE, levels=3 cycles=v
        Cycles per PCApply=1
        Using externally compute Galerkin coarse grid matrices
        GAMG specific options
          Threshold for dropping small values in graph on each level = 0.05  0.  0.
          ...
          Complexity:    grid = 1.05401
    Coarse grid solver -- level ------------------------------
      ...
      PC Object: (mg_coarse_) 4 MPI processes
        type: bjacobi
          number of blocks = 4
          ...
```

LISTING 3. Sample output of -snes_view.

PETSc provides flexible monitors to be used with solver views, which print the residual or function norm at each iteration of an iterative solver so that users can check the convergence of the solver and compare different algorithms. Listing 4 shows the output of nonlinear and linear solver monitors.

In summary, PETSc offers a wide set of complementary options to aid *debugging by e-mail* with the following common themes: users can enable debugging regardless of their computing environment; errors appear as early as possible; and the output is printed in well-formatted plain text for copy-and-paste or file attachments to e-mails and GitLab issues.

CONCLUSION

The increased prominence of data science and the transition to computing architecture heterogeneity require more, not less, high-quality numerical simulation and analysis software. This software is often created in community open-source environments; the

```
  0 SNES Function norm 1.223958326481e+02
    0 KSP Residual norm 1.223958326481e+02
    1 KSP Residual norm 2.137523917735e+01
    2 KSP Residual norm 4.364326343132e+00
    ...
   12 KSP Residual norm 3.922463112223e-10
  Linear solve converged due to CONVERGED_RTOL iterations 12
  1 SNES Function norm 3.922318262147e-10
Nonlinear solve converged due to CONVERGED_FNORM_RELATIVE iterations 1
```

LISTING 4. Sample output of -snes_monitor -ksp_monitor.

communities are crucial to the utility of such software. We have outlined some aspects of the open-source PETSc community and its collaboration strategies. Most of what was discussed apply to other numerical software communities. We concluded by focusing on mechanisms we use to allow community members to efficiently help one another at a distance using straightforward communication channels. The science and engineering of scientific software communities are only just beginning, and this topic is starting to receive more consideration at institutional levels. By sharing some of the PETSc community approaches, we hope to contribute to the wider scientific computing community as it seeks to improve the software programming process.

ACKNOWLEDGMENTS

The authors thank all PETSc users and developers for their many software, organizational, and conceptual contributions to the community. This material was based upon work funded in part by the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. This work was supported in part by the Exascale Computing Project under Grant 17-SC-20-SC, in part by a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award Number DE-SC0016140. The work of Matt Knepley and Jed Brown were supported in part by U.S. DOE Contract DE-AC02-0000011838.

REFERENCES

1. G. Avelino, E. Constantinou, M. T. Valente, and A. Serebrenik, "On the abandonment and survival of open source projects: An empirical investigation," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2019, pp. 1–12.
2. S. Balay *et al.*, "PETSc/TAO users manual," Argonne Nat. Lab., Lemont, IL, USA, Tech. Rep. ANL-21/39, Revision 3.16, 2021. [Online]. Available: https://petsc.org/
3. W. Bangerth and T. Heister, "What makes computational open source software libraries successful?," *Comput. Sci. Discov.*, vol. 6, no. 1, 2013, Art. no. 015010, doi: 10.1088/1749-4699/6/1/015010.
4. E. Bueler, *PETSc for Partial Differential Equations: Numerical Solutions in C and Python*. Philadelphia, PA, USA: SIAM, 2020, doi: 10.1137/1.9781611976311.
5. N. Eghbal, "Roads and Bridges: The unseen labor behind our digital infrastructure," 2016. [Online]. Available: https://www.fordfoundation.org/work/learning/research-reports/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure/
6. L. C. McInnes, M. A. Erik, W. Heroux, A. Draeger, S. S. Coghlan, and K. Antypas, "How community software ecosystems can unlock the potential of exascale computing," *Nat. Comput. Sci.*, vol. 1, pp. 92–94, 2021, doi: 10.1038/s43588-021-00033-y.
7. R. T. Mills *et al.*, "Toward performance-portable PETSc for GPU-based exascale systems," *Parallel Comput.*, vol. 108, 2021, Art. no. 102831, doi: 10.1016/j.parco.2021.102831.
8. "Request for information on the stewardship of software for scientific and high-performance computing," U.S. Dept. Energy, Office Adv. Sci. Comput. Res., Oct. 2021. [Online]. Available: https://www.govinfo.gov/app/details/FR-2021-10-29/2021-23582
9. M. J. Turk, "How to scale a code in the human dimension," in *Proc. Conf. Extreme Sci. Eng. Discov. Environ.: Gateway Discov.*, 2013, pp. 1–7, doi: 10.1145/2484762.2484782.
10. "Workshop on the science of scientific-software development and use," U.S. Dept. Energy, Office Adv. Sci. Comput. Res., Dec. 2021. [Online]. Available: https://www.orau.gov/SSSDU2021
11. J. Zhang *et al.*, "The PetscSF scalable communication layer," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 4, pp. 842–853, Apr. 2022, doi: 10.1109/TPDS.2021.3084070.

MARK ADAMS is a staff scientist in the Scalable Solvers Group at Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA. Adams received his Ph.D. degree in civil engineering from the University of California at Berkeley. Contact him at mfadams@lbl.gov.

SATISH BALAY is a software engineer at Argonne National Laboratory, Lemont, IL, 60439, USA. Balay received his M.S. degree in computer science from Old Dominion University. Contact him at balay@mcs.anl.gov.

OANA MARIN is a numerical analyst at Argonne National Laboratory, Lemont, IL, 60439, USA. Marin received her Ph.D. degree applied mathematics from the Royal Institute of Technology. Contact her at oanam@anl.gov.

LOIS CURFMAN MCINNES is a senior computational scientist at Argonne National Laboratory, Lemont, IL, 60439, USA. McInnes received her Ph.D. degree in applied mathematics from the University of Virginia. Contact her at curfman@anl.gov.

RICHARD TRAN MILLS is a computational scientist at Argonne National Laboratory, Lemont, IL, 60439, USA. Mills received his Ph.D. degree in computer science from the College of William and Mary. He is a member of IEEE. Contact him at rtmills@anl.gov.

TODD MUNSON is a senior computational scientist at Argonne National Laboratory, Lemont, IL, 60439, USA. He is a member of IEEE. Munson received his Ph.D. degree in computer science from the University of Wisconsin at Madison. Contact him at tmunson@anl.gov.

HONG ZHANG is an assistant computational mathematician at Argonne National Laboratory, Lemont, IL, 60439, USA. Zhang received his Ph.D. degree in computer science from Virginia Tech. Contact him at hongzhang@anl.gov.

JUNCHAO ZHANG is a software developer at Argonne National Laboratory, Lemont, IL, 60439, USA. Zhang received his Ph.D. degree in computer science from ICT, Chinese Academy of Science. Contact him at jczhang@anl.gov.

JED BROWN is an assistant professor at the University of Colorado at Boulder, Boulder, CO, 80309, USA. Brown received his Dr.Sc. degree in civil and environmental engineering from ETH, Zurich, Switzerland. Contact him at jed@jedbrown.org.

VICTOR EIJKHOUT is a research scientist at the Texas Advanced Computing Center of The University of Texas at Austin, Austin, TX, 78712, USA. Eijkhout received his Ph.D. degree in mathematics from Radboud University, The Netherlands. Contact him at eijkhout@tacc.utexas.edu.

JACOB FAIBUSSOWITSCH is a graduate research student at the University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA. Faibussowitsch received his B.Sc. degree in engineering mechanics from the University of Illinois at Urbana-Champagne. Contact him at faibuss2@illinois.edu.

MATTHEW KNEPLEY is an associate professor at the University of New York at Buffalo, Buffalo, NY, 14260, USA. Kenpley received his Ph.D. degree in computer science from Purdue University. Contact him at knepley@gmail.com.

FANDE KONG is a computational scientist and software developer at Idaho National Laboratory, Idaho Falls, ID, 83415, USA. Kong received his Ph.D. degree in computer science from the University of Colorado Boulder. Contact him at fande.kong@inl.gov.

SCOTT KRUGER is a scientist/VP with Tech-X Corporation, Boulder, CO, 80303, USA. Kruger received his Ph.D. degree in nuclear engineering and engineering physics from the University of Wisconsin-Madison. Contact him at kruger@txcorp.com.

PATRICK SANAN is a postdoctoral researcher at ETH Zurich, 8092, Zürich, Switzerland and an assistant computational mathematician at Argonne National Laboratory, Lemont, IL, 60439, USA. Sanan received his Ph.D. degree in applied and computational mathematics from the California Institute of Technology. Contact him at psanan@anl.gov.

BARRY F. SMITH is a senior research scientist at the Flatiron Institute, New York, NY, 10010, USA. Smith received his Ph.D. degree in mathematics from New York University. Contact him at bsmith@flatironinstitute.org.

HONG ZHANG is a research professor of computer science at the Illinois Institute of Technology, Chicago, IL, 60616, USA. Zhang received his Ph.D. degree in applied mathematics from Michigan State University. Contact her at hzhang@mcs.anl.gov.