The PETSc software package \[1, 2\] implements hundreds of published numerical algorithms and can use over 50 optional external software packages. PETSc is used by a broad range of scientific applications. This white paper addresses questions posed by the HPCOR workshop organizers.

**How many developers are involved and how is the development structured?** The PETSc team consists of about 10 core developers, most of whom are at the postdoctoral level and are located at various institutions worldwide. In addition to contributions by core developers, over 50 community contributors provide input, ranging from small patches to larger pieces of functionality, which is incorporated into PETSc each year.

**Language.** The primary language used for PETSc is C, with support for Fortran, Python, and Julia.

**Lines of code?** PETSc consists of roughly 400,000 lines of code. These include comments in source files but not source code in examples and tests.

**Primary methods?** Core PETSc capabilities are scalable algebraic solvers for ODEs/DAEs, nonlinear solvers, and linear solvers, including domain decomposition and multigrid.

**Types of problems/domains/science application problems?** PETSc is typically used in simulations involving partial differential equations as well as large-scale optimization problems.

**Scale of resources commonly used for production runs?** PETSc can be run at any scale: from a laptop to a cluster to the largest resources available at ALCF, NERSC, and OLCF.

**Supercomputers regularly used?** Most instances of PETSc on supercomputers are run by our users. User feedback suggests regular use on Edison, Hopper, Mira, Sequoia, Stampede, and Titan.

**Libraries/tools for prototyping?** PETSc is developed using standard free open source development tools, such as GNU Emacs, Git, and Valgrind.

**Libraries/tools for production science campaigns?** PETSc is itself a library for production science campaigns for a variety of domains, including materials science, nuclear engineering, subsurface flow, and magnetic fusion energy.

**Describe efforts to develop code (application, library, etc.) portable across diverse architectures.** The performance of PETSc depends primarily on good use of memory bandwidth, which is typically less sensitive to architectural differences than applications limited by the rate of floating point operations. We evaluated the use of threads in addition to MPI, but our performance evaluations have not shown any advantage over a pure (“flat”) MPI approach. Consequently, threading support in PETSc has been reduced to offer facilities for interfacing multithreaded user code with PETSc.\[1\]

---

\[1\] See companion whitepaper.
Where were the abstractions? PETSc employs data encapsulation via object oriented techniques. Memory encapsulation is achieved via MPI, which means that each process (stream of control) has by default access to only its memory regions; any sharing of memory must be explicitly enabled. This encapsulation model is different from thread parallelism, where by default all threads have access to all memory in a common memory region.

How much code re-use was possible? If something was not possible, please describe why. Code reuse is achieved by providing composable components in PETSc and by not relying on excessive code generation through the compiler, as would be the case with extensive C++ template techniques. This approach results in relatively small binaries, which are in turn more cache-friendly.

What successes have you had with performant code across difference architectures? Application codes based on PETSc perform well on a broad range of HPC architectures. PETSc’s approach to composable solvers enables users to investigate the design space of composable linear, nonlinear, and timestepping solvers for complex multilevel, multidomain, multirate, and multiphysics problems, without making premature choices about algorithms and data structures. Thus, at runtime users can select particular data structures and algorithms to exploit problem-specific knowledge and machine-specific characteristics.

Were the same algorithms applicable at all across the architectures? In many cases, higher-level algorithms apply across various architectures, while lower-level data structures may be customized.

What approaches did you reject and why? Because of the object-oriented nature of PETSc, experiments with C++ have been made, but we ultimately switched back to C because of flexibility and maintainability reasons. Also, experiments with threads (OpenMP, pthread) have not shown any performance gains over a flat MPI model. In the interest of a maintainable code base, thread-related code has been reduced to a minimum.

What is your greatest fear going to exascale for application portability and functionality? The PETSc team has no fear of going to exascale, just as we have had no fear of going to petascale in the past.

Acknowledgments
The authors were supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract DE-AC02-06CH11357.

References


Government License. The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.