From *SIAM News*, Volume 44, Number 7, September 2011

[HPC]^3: Making Scientific Codes More Powerful and Efficient

By David I. Ketcheson, Aron Ahmadia, and Matthew G. Knepley

Advancing science through computation requires a combination of knowledge from many disciplines, including numerical analysis, software and computer engineering, and physical knowledge of the application domain. Unfortunately, this knowledge is found in separate silos in the traditional academic world. Remarkable progress can be made when expertise from different disciplines is assembled for focused study in a single place. Recently, researchers from around the world, many from the SIAM community, converged on the King Abdullah University of Science and Technology in Thu'wal, Saudi Arabia, for just this purpose. The occasion was a KAUST-hosted workshop, "High Performance Computing and Hybrid Programming Concepts for Hyperbolic PDE Codes," or [HPC]^3.

Hybrid Programming Concepts

Modern scientific codes often comprise multiple levels written in different languages. The idea is that high-level languages (like Python) are better suited to express abstract ideas and organization, while low-level languages (like Fortran, C, or CUDA) can be used for the computational "kernels" (bottlenecks) to achieve high performance. High-level languages can leverage existing scientific libraries without forcing them to interface directly, by providing each with its own high-level interface.

This multilingual or "hybrid" programming approach has been applied successfully in the scientific Python community. With the advent of Python "wrappers" for libraries like PETSc (petsc4py) and Trilinos (pyTrilinos), codes written purely in Python can use the libraries. The wrappers interface directly to numpy, the numerical array package in Python supported by Enthought. One relatively cheap and effective way to parallelize existing application codes written in Fortran or C is to create Python interfaces to them and then incorporate petsc4py or pyTrilinos. The serial legacy code can be used without modification, and the low-level code has no need to be aware of parallelism. Another promising approach now gaining momentum is the automatic generation of C or Fortran kernels by high-level code: Application scientists write code in their "native" languages; that code is translated, first into numerical methods and finally into computer code. Such hybrid approaches were a major focus of [HPC]^3, which featured presentations on PETSc, Lisandro Dalcin's petsc4py and mpi4py, Andy Terrel's Ignition, a code-generation tool that creates domain-specific languages, and PetClaw.

The PetClaw code is one realization of the hybrid programming strategy. It is based on PyClaw, a Python interface developed by Kyle Mandli to Randy LeVeque's purely serial Fortran code Clawpack. PyClaw is a rewrite of the more abstract parts of Clawpack in Python, along with automated wrapping of the computational kernels by a tool called f2py. PetClaw is a parallelization of PyClaw spearheaded by Amal Alghamdi. The primary data structure in PyClaw is a numpy array; petsc4py can be used to package these arrays into PETSc DAs, the parallel structured-grid data structure in PETSc. The resulting code is both simple and elegant, requiring fewer than 300 lines of parallel-aware Python code.

Challenges for High-performance Hyperbolic PDE Codes

Wave propagation applications are often multiscale in nature. Tsunami waves traveling across the ocean and supernova blasts traveling across a galaxy are just two examples in which the size of interesting features is tiny compared to the domain of interest. For this reason, adaptive mesh refinement is often essential for such applications; making AMR codes scale on supercomputers, however, remains a very difficult task. In a featured presentation on this topic at [HPC]^3, "Making Adaptive Meshes Run on 10,000+ Processors," Wolfgang Bangerth described the implementation of AMR in the deal.II library via Carsten Burstedde's p4est, a library for the management of adaptively refined parallel meshes.

A roundtable discussion following the talk focused on the relative merits of the two dominant data structures used in AMR codes: octrees and patches. In the patch-based approach, the grid is refined uniformly over rectangular regions, which can overlap in arbitrary ways. In the octree approach, every node of the grid is attached to a "parent" corresponding to a coarser grid in recursive fashion. Comparison of the two approaches reveals non-trivial trade-offs in flexibility, efficiency, and software complexity. Most AMR codes use custom implementations of AMR that can be tightly coupled to other algorithmic aspects. The developers of the p4est library intend to allow codes to adopt AMR in a modular way.

Wave propagation problems often involve widely differing time scales as well, necessitating the use of implicit integration methods. Stable and robust implicit integration of highly nonlinear hyperbolic problems, still a significant challenge, was considered by David Keyes and Ravi Samtaney in back-to-back presentations at the workshop; the emphasis was on systems like magneto-hydrodynamics that feature waves with very different speeds. Most hyperbolic PDE codes focus on explicit time stepping, and those that feature implicit stepping often reimplement the required solvers.

By interfacing with PETSc, the PetClaw code provides the opportunity to apply a wide range of powerful and reliable solvers to hyperbolic PDEs. A group of workshop participants led by Jed Brown realized this potential. Incorporating semi- and fully implicit solvers through the PETSc time-stepping interface, they achieved good accuracy at a Courant number of 50 with only a few Newton iterates. Meanwhile, a group led by Matthew Emmett used PyWENO to generate high-order (up to 17) weighted essentially non-oscillatory (WENO) kernels for PetClaw, in both C and Fortran. Because of the ease of integrating multiple packages with Python-based interfaces, both of these efforts succeeded in just a few days.

Further efforts begun at the workshop continue to bear fruit, including work on implementation of GPGPU Riemann solvers (led by Jeff Stuart) and on automated generation of Riemann solvers based on high-level mathematical descriptions (led by Andy Terrel). Like the other workshop projects, these investigations rely on interfaces between high- and low-level languages to make scientific codes more powerful and efficient.

The [HPC]^3 workshop had the dual purpose of developing the PetClaw package and bringing together a diverse group of world-class scientists to push the boundaries of scientific computing in light of new techniques and paradigms for hybrid programming. Judging from the thought-provoking talks, flurry of coding, and lively discussions, it seems to have succeeded on both fronts.

Resources

[HPC]^3 workshop talks: https://sites.google.com/site/hpc3atkaust/archive-of-presenters-slides-pdfs.
Clawpack: http://www.clawpack.org.
PETSc: http://www.mcs.anl.gov/petsc/.
PetClaw/PyClaw: http://web.kaust.edu.sa/faculty/davidketcheson/pyclaw/.
petsc4py: http://code.google.com/p/petsc4py/.
Ignition: http://andy.terrel.us/ignition/.
PyWENO: http://memmett.github.com/PyWENO/.
p4est: http://www.p4est.org/.
deal.II: http://www.dealii.org/.

David Ketcheson is an assistant professor of applied mathematics and computational science in the Division of Mathematical and Computer Sciences and Engineering at KAUST. Aron Ahmadia is a postdoctoral fellow in the Supercomputing Laboratory at KAUST. Matthew Knepley is a senior research associate at the Computation Institute, University of Chicago.