PLUG-AND-PLAY CLUSTER COMPUTING: HIGH-PERFORMANCE COMPUTING FOR THE MAINSTREAM

To achieve accessible computational power for their research goals, the authors developed the tools to build easy-to-use, numerically intensive parallel computing clusters using the Macintosh platform. Their approach enables the user, without expertise in the operating system, to develop and run parallel code efficiently, maximizing the advancement of scientific research.

Accessible computing power has become the main motivation for cluster computing—some wish to tap the proliferation of desktop computers, while others seek clustering because they find access to large supercomputing centers to be difficult or unattainable. Both want to combine smaller machines to provide sufficient access to computational power. In this article, we describe our approach to cluster computing to best achieve these goals for scientific users and, ultimately, for the mainstream end user.

One approach, introduced in the mid-1990s, used a parallel computing message-passing library with the Linux operating system and became known as Beowulf-style cluster computing.[1] Today, the message-passing interface (MPI)[2] is a dominant industry standard, and many MPI implementations are available under open-source license. (Most major supercomputing centers only use MPI for distributed-memory parallel computing; the absence of other message-passing schemes on new hardware is evident at http://hpcf.nersc.gov/software/libs and www.npaci.edu/BlueHorizon/guide/ref.html.)

Our goal is to minimize the time needed to assemble and run a working cluster. Simplicity and straightforwardness are just as important as processing power because power provides nothing if it can't be used effectively. Moreover, our solution should provide a better total price-to-performance ratio and a higher commitment to the original purpose of such systems: providing the user with large amounts of accessible computing power.

Since 1998, we in the University of California, Los Angeles, Plasma Physics Group have been developing and using a solution to meet these design criteria. It's based on the Macintosh operating system and uses PowerPC-based Macintosh (Power Mac) hardware; we call it a Mac cluster.[3] In our ongoing effort to improve user experience, we continue to streamline the software and add numerous new features. With OS X, the latest, Unix-based version of the Mac OS (www.apple.com/macosx), we're seeing the convergence of the best of Unix with the best of the Mac.

We've extended the Mac's ease of use to parallel computing for the sake of accessible computing power. In this article, we describe how a user can build a Mac cluster and demonstrate how to operate it. We then describe the technology we built to

1521-9615/05/$20.00 © 2005 IEEE
Copublished by the IEEE CS and the AIP

DEAN E. DAUGER
Dauger Research
VIKTOR K. DECYK
University of California, Los Angeles

accomplish our goals. Part of our effort involves rethinking and streamlining cluster design, installation, and operation. We believe this experience has led us to a cluster solution that maximizes the user's accessibility to computational power.

Building, Running, and Debugging the Cluster

Streamlining cluster setup to the bare minimum, the steps to building a Mac cluster essentially involve connecting the computers to the network, assigning network names and addresses to the nodes, and installing the software.

Building a Mac cluster begins by collecting the hardware: the Power Mac G5s, one category 5 Ethernet cable with RJ-45 jacks per Mac, and an Ethernet switch. The latest Power Mac models have either fast (100BaseT) or Gbit Ethernet, so a switch of either type will function well. For each Mac, one end of a cable plugs into the Ethernet jack on the Mac and the other end to a port on the switch. System software is a simple matter: Macs come preinstalled with Mac OS X, so configuring the Macs generally involves making sure each one has a working Internet or IP connection and a unique name, specified in the Network and Sharing System Preferences. Finally, a software package called Pooch operates the cluster (see http://daugerresearch.com/pooch/ for a download version). Running the installer on each Mac's hard drive completes the parallel computer. Software installation on a node takes only a few seconds, brevity not found in other cluster types.

Because the intention is that the cluster user will spend most of his or her time interacting with the cluster performing job-launching activities, we've invested considerable effort in refining the interface design to minimize the time for the user to run a parallel job. In our documentation, we recommend that users first test their Mac cluster with a simple, reliable parallel computing job. For the purpose of this initial test, the AltiVec Fractal Carbon demo, a demonstration parallel application, is available for free download at http://daugerresearch.com/pooch/. This demonstration of high-performance computing also runs on a single node. The user runs the application in parallel by selecting New Job from Pooch's File menu. This action opens up a new Job window; the user can drag the AltiVec Fractal Carbon demo from the Finder to this Job window, as depicted in Figure 1.

Next, the user chooses nodes to run in parallel. By default, Pooch selects the node where the job is specified. To add more, the user clicks on Select Nodes…, which invokes a Network Scan window,

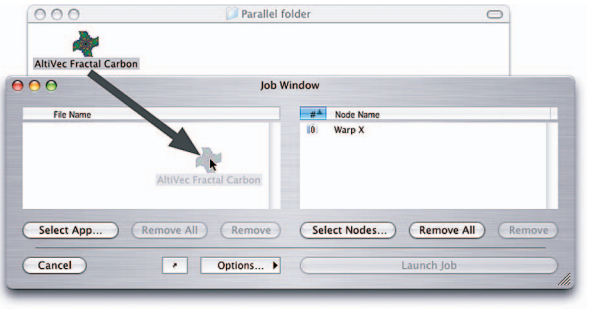

Figure 1. Using the cluster. To set up a parallel computing job, the user drags a parallel application, in this case the AltiVec Fractal Carbon demo, and drops it in Pooch's Job window.

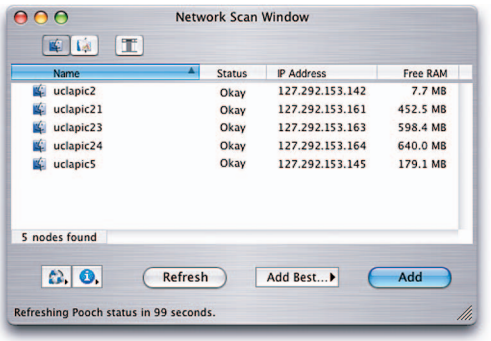

Figure 2. Selecting nodes. The user chooses computational resources via the Network Scan window, invoked by clicking on Select Nodes from the window in Figure 1.

shown in Figure 2. Double-clicking on a node moves it to the Job window's node list. If a machine running OS X has two processors, Pooch can use them as if they were separate nodes. Finally, the user starts the parallel job by clicking on Launch Job. Pooch should now be distributing copies of the parallel application to the other nodes and initiating them in parallel. Upon completion of its computational task, the program then calculates its achieved performance, which should be significantly greater than single-node performance.

This initial test also trains the user to accomplish the fundamental tasks required to run a parallel job: selecting an executable, selecting computational resources, and combining these selections through job initiation. Streamlining this user interface is important because submitting jobs is a repetitive task that can potentially occupy much of the user's time. We chose a GUI because it tolerates the type of error and imprecision that users can accidentally introduce when operating a device. This use of a

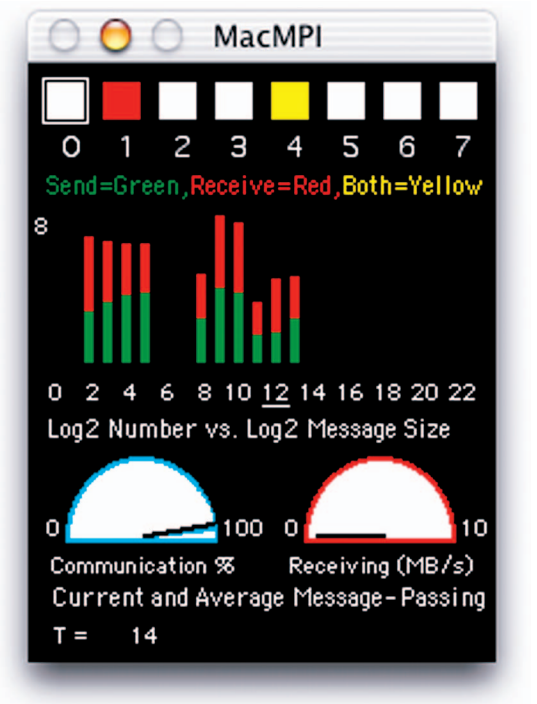

Figure 3. MPI modifications. The MacMPI monitor window keeps track of statistics about the execution of parallel applications.

GUI is meant to contribute to the efficiency with which the user can operate the cluster.

So that our group's researchers could maximize their time studying physics, we've added enhancements, beyond basic message passing, to the MPI implementation we call MacMPI that make it easier for them to develop parallel programs. One of these is the monitoring of MPI messages, controlled by a monitor flag in MacMPI: it can log every message sent or received. In its default setting, a small monitor window appears (see Figure 3). In this window, status lights indicate whether the node whose screen is being examined is sending or receiving messages from any other node. Green indicates sending, red indicates receiving, and yellow means both. Because messages are normally sent very quickly, these lights blink rapidly. However, if a deadlock occurs, a common occurrence for beginning programmers, the lights will stay lit. The moment such a problem occurs, though, a particular color pattern is immediately visible to the user, who can then apply the new information to debugging the code.

The monitor window also shows a color-coded histogram of the message sizes sent or received to draw the user's attention to the length of the messages the code is sending. The two dials in MacMPI's monitor window show the approximate percent of time spent in communication and the average and instantaneous speeds achieved during communication. Although approximate, these indicators are invaluable in revealing problems in the code and network.

Implementation

In the Mac cluster's design, we made the responsibilities of the communications library distinct and separate from the code that launches jobs and manages the cluster.

MacMPI

MacMPI, freely available from the AppleSeed site (http://exodus.physics.ucla.edu/appleseed/), is Viktor Decyk's 45-routine subset of MPI implemented via the Mac OS's networking APIs. It exists in two forms: MacMPI_X, which uses Apple's latest Open Transport implementation of TCP/IP available in both OS 9 and OS X, and MacMPI_S, which uses the Unix socket implementation in OS X (see http://developer.apple.com/documentation/CoreFoundation/Networking-date.html).

Using MacMPI, we achieve excellent network performance comparable to other implementations. On OS X, we achieve near peak speed of 100BaseT for large messages. Apple's Power Mac G5 hardware also comes with built-in Gbit Ethernet ports. On current hardware using a crossover Ethernet cable, the Open Transport implementation's latency is longer than MacMPI_S's, but both achieve more than 100 Mbytes/s bandwidth.

MacMPI is a source-code library that users can integrate into their executables. It forms a wrapper library for MPI code written in Fortran and C that assumes that just the fundamental preinstalled operating system is present. MacMPI takes advantage of as much of the operating system as possible to minimize its size and complexity. We've used this library on hardware normally not designated for cluster operation and configured it in virtually every possible configuration.

The Pooch Application

Pooch is a parallel computing and cluster management tool that can organize a job's files into subdirectories on other nodes and retrieve files on those nodes containing output from completed jobs. It can queue jobs and launch them only when certain conditions are met. It also has the ability to kill running, launching, or queued jobs. It can also exploit machines on which no one is logged in.

Pooch supports a wide variety of parallel programming environments, enabled by the convergence of technologies in OS X: Carbon, Cocoa,

Mach-O, Unix shell scripts, and AppleScripts. As of this writing, Pooch supports five different MPIs: MacMPI, mpich, MPI/Pro, mpich-gm (for Myrinet hardware), and LAM/MPI (see http://daugerresearch.com/pooch/mpi.html).

Pooch also features four user interfaces. In addition to its drag-and-drop GUI, Pooch's AppleScript interface makes it possible to write automatic and interactive scripts that perform customized job queuing and other cluster operations. A suite of command-line utilities makes it easy for users to log in from other platforms and control cluster operations.

Additionally, by sending commands through interapplication messages called AppleEvents, other applications can directly control Pooch to perform Grid-like behavior. The Fresnel Diffraction Explorer (FDE), an optics parallel desktop application, can initiate its own utilization of a local cluster via Pooch. Although the current incarnation of Globus (www.globus.org) and related technologies combine resources on a supercomputer level, our technology combines desktop machines. Unlike Globus and Condor (www.cs.wisc.edu/condor/), these features are installed, configured, and run by using an accessible GUI. With only a menu selection, desktop applications such as FDE today can automatically take advantage of resources elsewhere on the cluster. Such powerful yet easy-to-use features are essential for parallel computing to become mainstream.

Mac cluster security consists of administrative domains. Each copy of Pooch is hard-coded to a 512-bit encryption key that rotates for each encoded command sent and received; the way it rotates is unique to a particular user or group of Mac cluster users. This approach is much like how a research group shares access to office resources, such as a network printer or a copy machine. The cluster can similarly be shared, which enables it to operate independently of password databases or other external security systems. At the same time, the convenience for users is that the cluster is accessible when they open Pooch, making it very easy for them to use the cluster. Pooch makes little distinction between users and administrators because current administrative needs are so minor. A recently introduced bifurcation called Pooch Pro introduces user login and hierarchy, CPU quotas, and other supercomputer-like administrative features to Mac clustering.

Pooch can also include nodes on almost any subnet of the Internet. We've exercised that capability many times, either initiating jobs on our cluster from home or by combining nodes at arbitrary distances. We've even used Pooch to combine nodes at UCLA with machines in Munich, Germany, 10,000 km away.

Distinctions from Other Implementations

Several fundamental differences exist between our approach to cluster computing and that of others.

Division of API and launching mechanism. A fundamental difference from most other cluster types is the clear distinction and separation between the code that performs internode communications for the job and the code that performs job initiation and other cluster management. In most MPI implementations, such as mpich and LAM, these tasks are merged in one package. Only recently has work begun on versions whose organization identifies distinctions between these tasks, such as the emerging MPICH2 rewrite of mpich (http://www-unix.mcs.anl.gov/mpi/mpich2/).

No modification to the operating system. Making no modifications to the operating system let us simplify much of our software design. We didn't even add a runtime-linked library on the system, much less the system-level or even kernel-level modifications other cluster implementations make. We took this approach so that parallel executables could run on any node regardless of such modifications. We added as little as possible to the system by adding only one extra piece of executable code, Pooch, to run and operate the cluster. This approach keeps installation time to a minimum, which helps satisfy our design goals with regards to cluster setup.

No static data. All Pooch operations exclusively use dynamically determined information. Pooch, therefore, doesn't require an administrator to maintain any static data files about the cluster. On OS X 10.2 and later, Pooch's node discovery implementation uses the Service Location Protocol and Apple's Rendezvous (also called Zero-Conf, www.zeroconf.org) simultaneously (http://developer.apple.com/macosx/rendezvous). These TCP/IP-based discovery services provide Pooch with the information needed by MacMPI to organize and set up the internode connections for the parallel job.

Minimal assumptions about configuration. The absence of further configuration details about the cluster expresses how reliably it tolerates variations in configuration while interfacing and operating with hardware and software. The hardware need not be identical, command-line login isn't needed, network interfaces can vary, and computing hardware can differ. Again, as far

as the platform is concerned, Pooch and MacMPI are categorized as just application code. When we run demonstrations, we often ask the audience to volunteer their computers to add to the Mac cluster. It still runs despite the wide variety in configuration of these volunteered machines. This design has many implications for the mainstream because most end users don't want to worry about such details.

Minimal centralization. A common philosophy used to increase the performance of parallel codes is to eliminate bottlenecks. We've extended this concept to the Mac cluster by minimizing centralization such as single points of failure or other bottlenecks. Many Linux clusters assume some sort of shared storage is fully configured and operational before distributing executables or data files, but this is a well-known point of failure. Our cluster approach doesn't use shared storage, thus eliminating the potential for a bottleneck. Finally, Mac clusters don't have a head node, at least not in the sense of the head node found in typical Linux-based clusters. There is no permanent controlling unit or units, so the behavior is much more peer to peer. A node is designated node zero for the duration of the job only, but any node can be node zero for the next job and more than one node can be designated node zero simultaneously. All nodes can act as temporary head nodes, a transient state that occurs only during the brief seconds of the launch process.

Real-World Experience

The Mac cluster's performance is excellent for certain classes of problems, mainly those in which communication is small compared to the calculation, yet the message packet size is large. In 2002, for example, Apple introduced the Xserve, a rack-mounted version of a Power Mac meant for server solutions. In collaboration with the Applied Cluster Computing Group at NASA's Jet Propulsion Laboratory, the AltiVec Fractal Carbon demo achieved over 217 Gflops on its 33-XServe dual-processor G4/1000 cluster. (For further details, see http://daugerresearch.com/fractaldemos/JPLXServes/JPLXServeClusterBenchmark.html.)

The University of Southern California gave our team the opportunity to run the Fractal demo and Pooch on 56 of its dual-processor Power Mac G4/533's plus 20 of its dual-processor Power Mac G4/450's, where we achieved more than 233 Gflops. (For further details, see http://daugerresearch.com/fractaldemos/USCCluster/USCMacClusterBenchmark.html.) Note that these machines were part of the Language Arts undergraduate computer lab and weren't meant for cluster work, yet they achieved supercomputer-level results. The latest version of our software operates on unused, logged-out machines of this sort of computer lab.

We built a new cluster, called the Dawson cluster, in 2004 out of 128 dual-processor Xserves using the PowerPC 970FX ("G5") processor. Connected via a gigabit network and managed by UCLA's Academic Technical Services, this cluster achieved 1.21 TFlops using the AltiVec Fractal benchmark and Pooch. (For further details, see http://daugerresearch.com/fractaldemos/DawsonCluster/DawsonCluster128Benchmark.html.) It achieved a similar result using Linpack, placing the Dawson cluster in the Top500 list. Our group is actively using the Dawson cluster for physics research, and we hope to extend the machine to 256 Xserves (512 processors) in the coming year.

As an example of performance specific to our physics research, we successfully ran a 127 million particle 3D electrostatic PIC simulation on a four-node Macintosh G4/1000 dual-processor cluster.[4,5] The total time was 17.6 seconds per time step, with a $128 \times 128 \times 256$ grid, and the cost of the machines was less than US$10,000. Just a decade ago, such calculations required the world's largest and most expensive supercomputers!

Pictured in Figure 4, our inexpensive and powerful cluster of Power Mac G3s, G4s, and G5s has become a valuable addition to our UCLA Plasma Physics Group. We use it to introduce new members to parallel computing and run large calculations for extended periods. The solution we've found is fairly unique in that half of the nodes aren't dedicated for parallel computing. We regularly purchase high-end Macs and devote them for computation, reassigning the older, slower Macs for individual (desktop) use and data storage. This reuse of Macs in the cluster makes for a very cost-effective solution that satisfies both our parallel computing and desktop computing needs.

In addition, the cluster's flexibility lets us redirect computational resources very quickly within our group, which is useful for unfunded research or exploratory projects so that we can better prepare for an official proposal later. If one investigator needs to meet a short deadline, he or she can ask the research group, borrow their desktop Macs, and combine them with the dedicated Macs for one large job or many smaller ones.

The cluster's presence has encouraged new members of our group and visitors to learn how to write portable parallel MPI programs, which they can run later on larger computers elsewhere. The

cluster also encourages a more interactive style of parallel programming, in contrast to the batch-oriented processing encouraged by most other cluster types. We can also display on desktop machines the results of calculations made elsewhere in the cluster, which lets us study a simulation part way through the calculation. Checking for mistakes early on saves a great deal of computation time that might otherwise be wasted.

Plasma Physics and Additional Applications

The particle-in-cell (PIC) codes we use in our group are also used in several other high-performance computing projects, such as modeling fusion reactors[6] and advanced accelerators.[7] Such projects require massively parallel computers, but we've found it very convenient to perform research projects on more modest and user-friendly parallel machines such as those in Mac clusters.

Recent calculations by James Kniep and Jean-Noel Leboeuf have concentrated on studying various mechanisms of turbulence suppression in devices such as the Electric Tokamak at UCLA[8] and the DIII-D tokamak at General Atomics.[9] The researchers involved in these projects use the Mac cluster for smaller problems when they need fast turnaround for quick scoping, as well as for the production calculations that 8-node subsets of our cluster can accommodate. Their results compare favorably to experimental observations of plasma microturbulence characteristics in DIII-D discharges.[10]

We've also been highly successful in applying the cluster's computational power to the classroom environment, but the experience serendipitously led to a plasma physics conference presentation. At UCLA, the Physics 260 course entitled "Exploring Plasmas Using Computer Models" is a computational plasma physics course in which students learn about plasmas and operate and analyze data from a plasma physics code. The professor assigned the cluster for the students' course work with Parsec, a 3D fully electromagnetic particle-in-cell code that John Tonge wrote for investigating the physics of plasma confinement in a levitated internal conductor device. Some of the runs contained as many as 50 million particles on up to a $256 \times 256 \times 128$ grid. Such assignments would have been impossible without the local cluster. The runs could be started at the end of class on one day, and the output could be retrieved for analysis at the next class meeting—an impossibility at supercomputing centers because of their large queues and runtime limitations. The student work even led to an academic contribution to the field.[11]

Recently, John Huelsenbeck (University of California, San Diego) and Fredrik Ronquist (Uppsala University) wrote and released pMrBayes, a parallel application that performs Bayesian estimates of phylogeny for biology. While discussing approaches for running their parallel code, they described the simplest method was "to use Dauger's program Pooch to control the jobs" (http://morphbank.ebc.uu.se/mrbayes3/).

Figure 4. A portion of our Mac cluster. The cluster in the Department of Physics at the University of California, Los Angeles, features a mix of Power Mac G5s and their predecessors. It routinely combines dedicated nodes with desktop machines to perform physics calculations.

Our approach is unique because, while other solutions seem to direct little, if any, attention to usability, tolerance to variations in configuration, and reliability outside tightly controlled conditions, we find such issues to be as important as raw performance. We believe the ultimate vision of parallel computing is technology so reliable and trivial to install, configure, and use that the user will barely be aware that computations are occurring in parallel.

We organize problems with using parallel computers into two categories: one, building, operating, managing, and maintaining such a machine, and two, determining how best to solve an application on that machine. We've described how our work solves the former so that users can maximize their energy on the latter. The simplicity of using a Mac cluster technology makes it a highly effective solu-

tion of all but the largest calculations, and we're continuing to improve on our work for the sake of those users and to respond to their feedback. We're partnering with industry and other entities to enable shrink-wrapped applications of clusters. Our next step is to expand the cluster to incorporate additional Grid-like and supercomputing features and further enhance its service to users.

Acknowledgments

Many people have provided us useful advice over the last few years. We acknowledge help given by Bedros Afeyan from Polymath Research; Ricardo Fonseca from Instituto Superior Técnico, Lisbon, Portugal; and Frank Tsung, John Tonge, and Warren Mori from the University of California, Los Angeles, and the Applied Cluster Computing Group at NASA's Jet Propulsion Laboratory.

References

1. T.L. Sterling et al., *How to Build a Beowulf*, MIT Press, 1999.
2. M. Snir et al., *MPI: The Complete Reference*, MIT Press, 1996.
3. V.K. Decyk, D. Dauger, and P. Kokelaar, "How to Build an AppleSeed: A Parallel Macintosh Cluster for Numerically Intensive Computing," *Physica Scripta*, vol. T84, 2000, pp. 85–88.
4. V.K. Decyk, "Benchmark Timings with Particle Plasma Simulation Codes," *Supercomputer 27*, vol. V-5, 1988, pp. 33–43.
5. V.K. Decyk, "Skeleton PIC Codes for Parallel Computers," *Computer Physics Comm.*, vol. 87, nos. 1 and 2, 1995, pp. 87–94.
6. R.D. Sydora, V.K. Decyk, and J.M. Dawson, "Fluctuation-Induced Heat Transport Results from a Large Global 3D Toroidal Particle Simulation Model," *Plasma Physics and Controlled Fusion*, vol. 38, no. 12A, 1996, pp. A281–A294.
7. K.-C. Tzeng, W.B. Mori, and T. Katsouleas, "Electron Beam Characteristics from Laser-Driven Wave Breaking," *Physical Rev. Letters*, vol. 79, no. 26, 1997, pp. 5258–5261.
8. M.W. Kissick et al., "Radial Electric Field Required to Suppress Ion Temperature Gradient Modes in the Electric Tokamak," *Physics of Plasmas*, vol. 6, no. 12, 1999, pp. 4722–4727.
9. J.C. Kniep, J.-N. Leboeuf, and V.K. Decyk, "Gyrokinetic Particle-In-Cell Calculations of Ion Temperature Gradient Driven Turbulence with Parallel Nonlinearity and Strong Flow Corrections," Poster 1E25, Int'l Sherwood Fusion Theory Conf., 2003, www.sherwoodtheory.org/sherwood03/agenda.html.
10. T.L. Rhodes et al., "Comparison of Turbulence Measurements from DIII-D L-Mode and High-Performance Plasmas to Turbulence Simulations and Models," *Physics of Plasmas*, vol. 9, no. 5, 2002, pp. 2141–2148.
11. J. Tonge et al., "Kinetic Simulations of the Stability of a Plasma Confined by the Magnetic Field of a Current Rod," *Physics of Plasmas*, vol. 10, no. 9, 2003, pp. 3475–3483.

Dean E. Dauger is president of Dauger Research. His technical interests include computational physics, particularly the dynamics of complex quantum systems, and high-performance computing and visualization. Dauger has a PhD in physics from the University of California, Los Angeles. He is a member of the APS and the IEEE Computer Society. Contact him at d@daugerresearch.com.

Viktor K. Decyk is a research physicist in the Department of Physics and Astronomy at the University of California, Los Angeles. His technical interests include computational plasma physics using parallel particle-in-cell methods. Decyk has a PhD in physics from the University of California, Los Angeles. He is a member of the APS, SIAM,ACM, and the IEEE Computer Society. Contact him at decyk@physics.ucla.edu.