Clúster de Cómputo “Enchufar y Listo”

“Plug-and-Play” Cluster Computing

Dean E. Dauger
Dauger Research, Inc.
http://daugerresearch.com/

Viktor K. Decyk
Department of Physics
University of California, Los Angeles
http://exodus.physics.ucla.edu/viktor/

Resumén

Debido a la alta demanda en poder computacional planteada por nuestras metas de investigación, hemos desarrollado varias herramientas para construir clústeres para computación numericamente intensiva con computadoras Macintosh conectadas en parallelo. Hemos hayado que la usabilidad y fiabilidad de la tecnología usada es tan importante como su eficacia. Nuestra estrategia es permitir al usuario, quien no tiene que tener mucha experiencia con el sistema operativo, el desarrollo y uso eficiente de código paralelizado, haciendo posible un óptimo avance en investigaciones científicas. Describimos en este artículo las decisiones de diseño que tomamos para obtener estas metas y damos muestras de las últimas aplicaciones de nuestras herramientas. Al "reinventar" el clúster, proveemos una singular solución diseñada para maximizar accesibilidad para los usuarios.

Abstract

To achieve accessible computational power for our research goals, we developed the tools to build numerically-intensive parallel computing clusters on the Macintosh platform. We find that the usability and reliability of the technology is as important as its performance. Our approach is designed to allow the user, without expertise in the operating system, to most efficiently develop and run parallel code, enabling the most effective advancement of scientific research. In this article we describe the design decisions we made to accomplish these goals and introduce the latest applications of our approach. By “reinventing” the cluster computer, we provide a unique solution designed to maximize accessibility for users. http://daugerresearch.com

I. Introduction

Accessible computing power, as a goal, is 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 difficult or unattainable. Both are led to ask how smaller machines can be combined to provide sufficient access to computational power. In this article, we describe how the design decisions made for our approach to cluster computing best achieves these goals for scientific users and, ultimately, for the mainstream end user. To show the potential of this clustering paradigm, we highlight some of its newest applications of this alternative approach.

One approach, introduced in the mid-1990’s, 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] has become a dominant industry standard [3], and many MPI implementations are available under open source license.

Beowulf, however, has taught us that the solution must be productive and cost-effective by requiring only a minimum of time and expertise to build and operate the parallel computer. Specifically, our goal is to minimize the time needed to assemble and run a working cluster. The simplicity and straightforwardness of this solution is just as important as its processing power because power provides nothing if it cannot be used effec-

tively. This solution would provide a better total price to performance ratio and a higher commitment to the original purpose of such systems: provide the user with large amounts of *accessible* computing power.

Since 1998, we have been developing and using a solution that meet those design criteria. Our solution is based on the Macintosh Operating System using Macintosh hardware; we call it a Mac cluster. [4] In our ongoing effort to improve the user experience, we continue to streamline the software and add numerous new features, adapting to the latest changes in the platform. With OS X, the latest, Unix-based version of the Mac OS, [5] we are seeing the convergence of the best of Unix with the best of the Mac. Also, we have responded to the recent adoption of Intel processors by extending our technology to support mixed clusters of Intel- and PowerPC-based Macs.

We have extended the Macintosh's famed ease-of-use to parallel computing. In the following, we describe how a user can build an Mac cluster and demonstrate how that user can operate it. We then describe technical details, as much as space allows, regarding important design choices we made to accomplish these design goals and the consequences of those choices, emphasizing how our solution is different from other cluster types. Part of our effort has been to rethink and streamline cluster design, installation, and operation. We believe these design principles have led us to a cluster solution that maximizes the user's accessibility to computational power. Finally, we show the potential of our solution by introducing new applications of our cluster technology.

II. The User's Cluster Experience

A. Building a Mac Cluster

Streamlining cluster setup to the bare minimum, the steps to building a Mac cluster have been distilled to connecting the computers to the network, assigning network names and addresses to the nodes, and quickly installing the software. The following paragraphs completely define the components and procedures for setting up a Mac cluster:

Building an Mac cluster begins by collecting the hardware: Power Macs or Mac Pros, one Category 5 Ethernet cable with RJ-45 jacks per Mac, and an Ethernet switch. Almost all the latest Mac models have Gigabit Ethernet, so a 100BaseT switch or faster with at least as many ports as there are Macs functions 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. Configuring the Macs generally involves making sure each Mac has an working Internet or IP connection and a unique name, specified in the Network and Sharing System Preferences.

Finally, a software package called Pooch is used to operate the cluster. A download version is available. [6] Running the installer on a hard drive of each Mac completes the parallel computer. Software installation on a node takes only a few seconds, a brevity not found in other cluster types.

B. Running a Mac Cluster

Because the intention is that the cluster user will spend most time interacting with the cluster performing such job launching activities, we have invested considerable effort refining the design of this user interface 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 Power Fractal app, a demonstration parallel application, is available for free download. [6] This demonstration of high-performance computing also runs on a single node.

The user runs this application in parallel by selecting New Job… from the File menu of Pooch. This action opens up a new Job Window. The user may drag Power Fractal from the Finder to this Job Window, depicted in Figure 1.

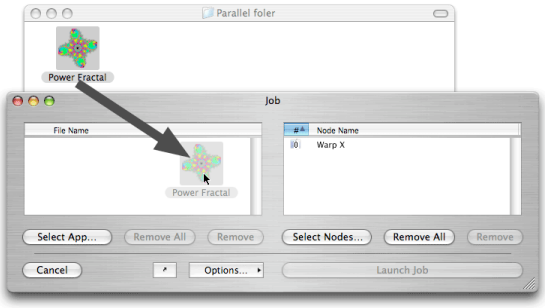

Figure 1. To set up a parallel computing job, the user drags a parallel application, in this case the Power Fractal, and drops it in the Job Window of Pooch.

Next, the user chooses nodes to run in parallel. By default, Pooch selects the node where the job is being specified. To add more, the user clicks on Select Nodes…, which invokes a Network Scan Window, shown in Figure 2.

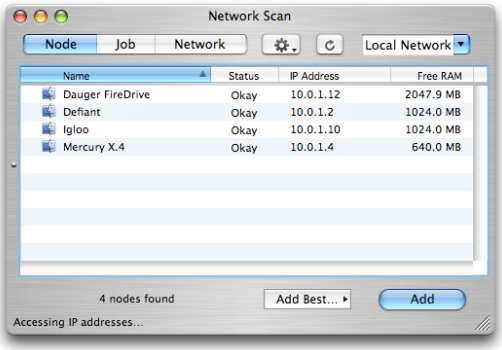

Figure 2. Selecting nodes is performed using the Network Scan Window, invoked by clicking on "Select Nodes…" from the window in the previous figure.

Double-clicking on a node moves it to the node list of the Job Window. If a machine running OS X has two processors, Pooch can use them as if they were separate nodes.

Finally, the parallel job must be started by clicking on Launch Job. Upon completion of its computational task, the demo 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. We have distilled the operation into three fundamental steps: 1. Selecting a executable; 2. Selecting computational resources; and 3. Combining these selections through job initiation. We consider the streamlining of this user interface to be important because submitting jobs is a repetitive task that potentially can occupy much of the user's time because of the intended high frequency of this task.

C. Debugging on a Mac Cluster

So that the Plasma group's physics researchers can maximize their time studying physics, we have 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, which can log every message sent or received. In its default setting, a small monitor window appears, shown in Figure 3. In this window, status lights indicate whether the node whose screen is being examined is sending and/or receiving messages from any other node. Green indicates sending, red indicates receiving, and yellow means both. Since messages normally are sent very fast, these lights blink rapidly. However, if a deadlock occurs, which is a common occurrence for beginning programmers, the lights will stay lit. The moment such a problem occurs, 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 similarly color-coded histogram of the size of messages being sent or received. The purpose of this histogram is 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. While approximate, those indicators have been invaluable in revealing problems in the code and the network.

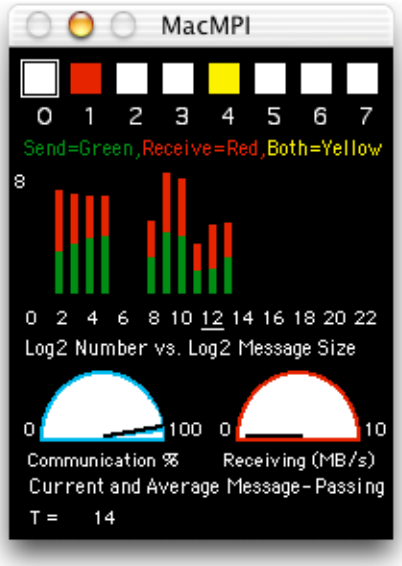

Figure 3. The monitor window of MacMPI, which keeps track of statistics about the execution of the running parallel application.

III. Design and Implementation

A. Division of API and Launching Utility

A fundamental difference from most other cluster types is the clear distinction and separation between the code that performs the 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 have incarnations emerged that identify distinctions between these tasks, such as the new MPICH2 rewrite of mpich. [7]

In the design of the Mac cluster, we made the responsibilities of the communications library distinct and separate from the code that launches the jobs and manages the cluster, a separation that has existed since the Mac cluster's inception in 1998. We call the former MacMPI, while the current incarnation of the latter is called Pooch.

MacMPI, freely available from the AppleSeed site at UCLA Physics, is Decyk's 45 routine subset of MPI implemented using the Mac OS networking APIs. It exists in two major types: the first, MacMPI_X, uses Apple's latest Open Transport implementation of TCP/IP available in both OS 9 and OS X while the second, MacMPI_S, uses the Unix sockets implementation in OS X. [8] Each has a corresponding "Universal Binary" derivative, MacMPI_XUB and MacMPI_SUB, to support running parallel applications on Intel- and PowerPC-based Macs simultaneously. Apple uses the "Universal" moniker to identify compiled code that runs natively on all current Mac processors. [9] MacMPI's performance results, comparable with the performance of open-source MPIs, are detailed further on our web site. [10]

Pooch is a parallel computing and cluster management tool designed to provide users accessibility to parallel computing. Pooch can organize the job's files into subdirectories on the other nodes and retrieve files on those nodes containing output from completed jobs. It can queue jobs and launch them only when certain conditions have been met. It also has the ability to kill running jobs, launching jobs, and queued jobs. It keeps track of these jobs and reports their status in an appealing GUI. It can also take advantage of machines where no user is logged in.

Pooch supports the widest variety of parallel programming environments, enabled by the convergence of technologies in OS X: Universal, Carbon, Cocoa, Mach-O, Unix shell scripts, or AppleScripts. [5] As of this writing, Pooch supports five different Message-Passing Interfaces (MPIs): MacMPI, mpich, MPI/Pro, mpich-gm (for Myrinet hardware), and LAM/MPI. [11] Because of OS X, MPIs of such varied histories are all now supported in the one environment. We are investigating further expansion of MPI support.

Pooch Pro, in its 2004 debut, introduced password-protected accounts with administrator and user hierarchy to Mac clusters. This bifurcation of Pooch identifies CPU time with particular users, making it possible to apply CPU time quotas, by job or by time period, to users' computing jobs. These can be used to regulate user access particular nodes CPU time not used in one time period can also be "rolled over" to the next, like cell phone services.

By sending commands through interapplication messages called AppleEvents, other applications can directly control Pooch to perform automated Grid-like behavior. The Fresnel Diffraction Explorer (FDE), an optics parallel desktop application, can initiate its own utilization of a local cluster via Pooch. While the present incarnation of Globus [12] and related technologies combine resources on a supercomputer level, our technology combines desktop machines. Unlike Globus and Condor [13], these features are installed, configured, and run 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 prerequisites for parallel computing to become mainstream. We have leveraged some of these features for new applications described below.

B. No Modification to the Operating System

Making no modifications to the operating system allowed us to simplify much of our software design. In our approach, we do not even add any runtime-linked library on the system, much less the system-level or even kernel-level modifications many cluster designs make. We took this approach so that parallel executables can run on any node regardless of such modifications.

Consequently, MacMPI is a source code library that users integrate into their executable. MacMPI is a wrapper library that assumes only the fundamental, preinstalled operating system is present and no more. MacMPI takes advantage of as much of the operating system as possible to minimize its size and complexity. We have utilized this library on hardware normally not designated for cluster operation and configured in virtually every possible configuration.

The operating system itself does not have cluster services. We add as little as possible to the system by adding only one additional 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 set up.

The design decision to stay above the operating system level is a lesson learned from earlier desktop computing. In the original Mac OS, extra tools nicknamed "INIT"s and "cdev"s, later known as "Extensions" and "Control Panels", respectively, inserted code into the operating system or replaced pieces of the operating system. Over the years, users would experience bugs, crashes, or other unreliability because of conflicts between the OS and the extensions or between extensions or later versions of the OS. This led to a then-thriving segment of the industry devoted to catching such conflicts. [14]

It was later determined, and encouraged by Apple, that the best way to add functionality to an operating system was using the application layers only. Such extensions and control panels were subsequently discouraged, and finally made virtually impossible to write in OS X. Likewise, our approach is to follow the rules that apply to the typical desktop application: use the documented, officially supported API and your code will be robust and backwards- and forwards-compatible.

C. Takes Advantage of a Consistently Supported API

At UCLA Physics, we do not have the resources to build or rebuild something as complex as an operating system or the APIs it provides to applications. Therefore, we took advantage of APIs that were already present and officially supported in the operating system.

The nature of this support is two-fold. First, Apple promised to support the Carbon API as explicitly documented, in these operating systems and all future releases of OS X. Second, if there is an inconsistency or other bug, Apple provides a mechanism called the Apple Bug Reporter with which any user of their operating system can report such issues with any supported release of their operating system or libraries. [15] A well-reproduced problem, described and submitted there, has historically been fixed in a timely manner. We are taking advantage of Apple's commercial, non-scientific motivation to provide a consistent, reliable, well-behaving API, operating system, and hardware. Our approach is to take advantage of such long-term motivations.

D. No Static Cluster Data

All Pooch operations exclusively use dynamically-determined information. Pooch, therefore, does not 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 Service Location Protocol and Apple's Bonjour (a.k.a. ZeroConf) simultaneously. [16] 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.

During the procedure of Section II B, above, Pooch uses encrypted connections to determine up-to-the-minute information about nodes, including their availability and capability, making it possible for the user to make decisions on which nodes to use and in what order. Pooch then provides network information to the MPI based on these decisions.

No assumptions have been made about particular hardware at particular addresses being available. By relying on dynamically determined network information, Pooch's GUI (see Figure 2) transparently encourages users to select hardware confirmed to be functioning and available for use.

We use an important concept here. A static node list could list nodes that are in fact non-functional, and a problem is discovered only when a job fails, which could at the outset be due to a variety of potential problems in any node in the list. By making dynamic discovery part of the node selection process, problem nodes are already eliminated before the user makes a decision. That design automatically eliminates a host of potential sources of failure that the user might encounter.

MacMPI assumes it is fed an arbitrary list of IP addresses to begin its work. That design decision makes it possible for Pooch to include nodes on almost any subnet of the Internet. We have exercised that capability many times, either initiating jobs on our cluster from home, or even combining nodes at arbitrary distances. Pooch has even been used to combine nodes at UCLA in Los Angeles, California, with machines in Munich, Germany, 10 000 km apart. Further details are in the documentation available with the distribution.

E. Minimum Assumptions about Configuration

Pooch makes as few assumptions as possible about the cluster configuration. The configuration requirements are that the node has a working network connection with a unique IP address and a unique network name. This configuration is considered minimal in today's personal computers, due to the ubiquity of web browser use and file exchange via the network. Again, our approach takes advantage of a preexisting condition in the computer industry.

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. The network interfaces can vary (100BaseT, 10BaseT, Gigabit, IrDA (infrared), Airport (wireless), FireWire (a. k. a. IEEE 1394)). Computing hardware can be different (Intel Cores of any speed, PowerPC G5s of any speed, multiple processors, desktops, portables, rack-mount Xserves). When we demonstrate the technology to others, we often ask the audience to volunteer their computers to add to the Mac cluster. The cluster runs despite the wide configuration variety of these volunteer machines. This design has great implications for the mainstream because end users do not wish to be concerned with such details.

The inexpensive and powerful cluster of Power Mac G4s and G5s has become a valuable addition to the UCLA Plasma Physics group. The solution at UCLA Physics is fairly unique in that half of the nodes are not dedicated for parallel computing. We purchase high-end Macs and devote them for computation while reassigning the older, slower Macs for individual (desktop) use and data storage. Thus, we are reusing the Macs in the cluster, making for a very cost-effective solution to satisfy both our parallel computing *and* desktop computing needs. The Mac cluster is unique in this regard, made possible by how tolerant the software is of variations in configuration. Because our software is now "Universal", we have already paved the way to adapt easily to mixing recently introduced Intel-based Mac Pros with our PowerPC hardware.

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

F. Minimum Centralization

A common strategy used to increase the performance of parallel codes is to eliminate bottlenecks. We have extended that concept to the Mac cluster by minimizing centralization such as single points of failure or other bottlenecks. This decision leads to certain design consequences.

Although many Linux clusters assume some sort of shared storage (NFS, AFS, etc.) be fully configured and operational in order to distribute executables or data files, no shared storage is needed for the Mac cluster because it is a potential single point of failure. Also, in a Mac cluster, there is no "head node", at least not in the sense of the "head node" of the typical Linux-based cluster. There is no permanent controlling unit or units. The traditional concept of "server" and "client" is not used.

Rather, we chose a decentralized approach. Operations, whenever possible, are performed much more like "peer to peer". All nodes can act as "temporary head nodes", a transient state occurring only during the brief seconds of the launch process. If a user finds that a node is down, that user can simply move on to another node for use in the next parallel job. Node identification by number is therefore a temporary state, enabling users to flexibly chose how to combine nodes for cluster computation from job to job.

IV. Plasma Physics, *Mathematica*, and More

The PIC codes at the UCLA Plasma Physics Group are used in a number of High-Performance Computing projects, such as modeling fusion reactors [17] and advanced accelerators [18]. For those projects massively parallel computers are required, but the group has found it very convenient to perform research projects on more modest and user-friendly parallel machines such the Macintosh clusters.

Simplifying the problem of building, operating, and maintaining a parallel cluster allows our group to use its cluster to focus on physics research. The Mac cluster at UCLA Physics is primarily used for plasma physics projects. The UCLA Parallel PIC Framework (UPIC) is a set of trusted components written in object-oriented Fortran90 that supports multiple plasma PIC codes. [19] Developed on the Mac cluster at UCLA Physics but scalable to the largest supercomputers, this framework supports multiple numerical methods, different physics approximations (both in 2D and 3D), and various numerical optimizations for different plasma conditions and scenarios. QuickPIC, a quasi-static code for studying plasma-based accelerators, utilizes coupled 3D and 2D PIC models implemented using UPIC. [20] 2D and 3D implementations of quantum PIC code, which uses a semiclassical approximation of Feynman path integrals to solve multiparticle quantum problems, also uses UPIC on the Mac cluster. [21] Performance of these codes on Power Mac G5s in comparison is on our web site. [10] Preliminary tests using Macs with the Intel Core show performance per "core" to be at least that of the G5. We will able to say more as we gain experience with the latest Mac Pro systems.

Our Mac cluster has been applied to numerous scientific applications, most thus far written in Fortran or C to MPI calls, and some are listed on our web site. [22] However, our intention is not to stop there. In partnership with Advanced Cluster Systems, [23] we announced (August 2006) running Wolfram Research's *Mathematica* [24] in parallel on a Mac cluster. [25] This "shrink-wrapped" commercial application was not designed to operate in parallel, so we created a way to launch and coordinate multiple Mathematica kernels on a cluster and added MPI calls to the Mathematica runtime environment to support node-to-node and collective messaging just like on massively parallel supercomputers. Because of certain peculiarities in the Mathematica language, like underscores being not allowed in function names, some adaptations and reinterpretations were required, but the principle ideas of the distributed-memory message-passing paradigm for parallel computing are now expressed in *Mathematica*. Using the AppleEvent interface described above, our toolkit taps the Mac cluster via Pooch, locates and launches licensed local *Mathematica* kernels, and then loads and configures the MPI libraries in the *Mathematica* runtime environment. The potential of combining the analytical power of *Mathematica* with the power of cluster computing is unprecedented. [26]

We believe the potential impact of combining Mac clustering with commercial technologies is staggering. Yet another part of our first step to make this vision real is to parallelize QuickTime compression using Mac clusters as well. [27] We just released (January 2007) version 1.0 of a QuickTime plug-in that intercepts data from commercial video editing applications, like Apple's iMovie, Final Cut Express, Final Cut Pro, and Motion, and redistributes the raw data for compression in parallel on a Mac cluster, then gathers and reorganizes the compressed data into QuickTime movies as output, shown in Figure 4. With the latest video codecs like H.264 taking many hours to compress each hour of source video, the clock time of such tasks can be substantial. This QuickTime plug-in automatically accesses the Mac cluster via Pooch as described for FDE above

and performs the task faster, reducing the clock time of such problems significantly and benefiting the end user. Making high-performance computing accessible to users via major commercial applications is part of our vision for plug-and-play cluster computing.

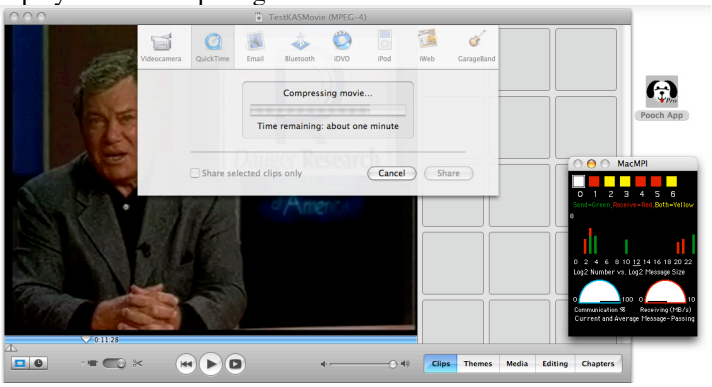

Figure 4. MacMPI being used while compressing video data from iMovie, a commercial video editing application, on a Mac cluster.

V. Conclusion

Our goal is to maximize the benefits of parallel computing for the end user. By assuming only the minimum configuration of the hardware and operating system, the Mac cluster design has the potential to provide an significant advantage to cluster users. The simplicity of using Mac cluster technology makes it a highly effective solution for all but the largest calculations. We are continuing to improve upon our work for the sake of those users and respond to their feedback.

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 (rather than merely to build raw processor power) to make the technology is so reliable and trivial to install, configure, and use that the user will barely be aware that computations are occurring in parallel. This article presents our progress in building and applying the "plug-and-play" technology to make that vision come true.

We organize problems with using parallel computers into two categories: 1. Building, operating, managing, and maintaining such a machine; and 2. Determining how best to solve an application on that machine. Above we describe how our work solves the former so that users can maximize their energy on the latter. Mac clusters' feature set is on par with other cluster types and is expanding to grid-like and supercomputing features. However, unlike other approaches, the Mac cluster solution makes the problem of building and operating a parallel computer easy and therefore enables the user to most efficiently write, debug, and run parallel codes. Because of their ability to make computational power accessible, Macintosh clusters are uniquely capable of maximizing the impact of parallel computing for scientific and mainstream users.

VI. Acknowledgments

Many people have provided us useful advice over the last few years. We thank J. Manuel Urrutia for translating the abstract. We acknowledge help given by Bedros Afeyan from Polymath Research, Inc., Ricardo Fonseca from IST, Lisbon, Portugal, Frank Tsung and John Tonge from UCLA, and the Applied Cluster Computing Group at NASA's Jet Propulsion Laboratory.

VII. References

[1] T. L. Sterling, J. Salmon, D. J. Becker, and D. F. Savarese, How to Build a Beowulf, [MIT Press, Cambridge, MA, USA, 1999].
[2] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, MPI: The Complete Reference [MIT Press, Cambridge, MA, 1996]; William Gropp, Ewing Lush, and Anthony Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface [MIT Press, Cambridge, MA, 1994].

[3] 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 NERSC: http://hpcf.nersc.gov/software/libs/ and at NPACI: http://www.npaci.edu/BlueHorizon/guide/ref.html
[4] V. K. Decyk, D. Dauger, and P. Kokelaar, "How to Build An AppleSeed: A Parallel Macintosh Cluster for Numerically Intensive Computing," Physica Scripta T84, 85, 2000.
[5] http://www.apple.com/macosx/
[6] http://daugerresearch.com/pooch/
[7] http://www-unix.mcs.anl.gov/mpi/mpich2/
[8] http://developer.apple.com/documentation/CoreFoundation/Networking-date.html
[9] http://www.apple.com/universal/
[10] See http://exodus.physics.ucla.edu/appleseed/
[11] See links at: http://daugerresearch.com/pooch/mpi.html
[12] http://www.globus.org/
[13] http://www.cs.wisc.edu/condor/
[14] A leading software product of that industry was "Conflict Catcher" published by Casady & Greene: http://www.casadyg.com/. That company closed in 2003.
[15] http://bugreport.apple.com/
[16] http://developer.apple.com/bonjour/ and http://www.zeroconf.org/
[17] 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.
[18] 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.
[19] V. K. Decyk and C. D. Norton, "UCLA Parallel PIC Framework", Computer Physics Communications 164 (2004) 80-85.
[20] C. K. Huang, V. K. Decyk, C. Ren, M. Zhou, W. Lu, W. B. Mori, J. H. Cooley, T. M Antonsen Jr., T. Katsouleas, "QuickPIC: A highly efficient particle-in-cell code for modeling wakefield acceleration in plasmas", *J. Comp. Phys.* Volume 217, Issue 2, 20 September 2006, 658-679.
[21] D. E. Dauger, V. K. Decyk, and J. M. Dawson, "Using semiclassical trajectories for the time-evolution of interacting quantum-mechanical systems", Journal of Computational Physics 209 (2005) 559-581.
[22] http://daugerresearch.com/pooch/user.shtml
[23] http://acs-grid.com/
[24] http://wolfram.com/
[25] http://daugerresearch.com/pr/poochmpimath.shtml
[26] http://daugerresearch.com/pooch/mathematica/
[27] http://daugerresearch.com/pooch/quicktime/