# "Plug-and-Play" Cluster Computing: HPC Designed for the Mainstream Scientist

Dean E. Dauger[1] and Viktor K. Decyk[2]

[1] Dauger Research, Inc., P. O. Box 3074, Huntington Beach, CA 92605 USA
http://daugerresearch.com/
[2] Department of Physics, University of California, Los Angeles, CA 90095 USA
http://exodus.physics.ucla.edu/

**Abstract.** At UCLA's Plasma Physics Group, to achieve accessible computational power for our research goals, we developed the tools to build numerically-intensive parallel computing clusters on the Macintosh platform. 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, in technical detail, the design decisions we made to accomplish these goals. We found it necessary for us to "reinvent" the cluster computer, creating a unique solution that maximizes accessibility for users. See: http://daugerresearch.com/

## 1 Introduction

Accessible computing power is becoming the main motivation for cluster computing. Beowulf [1], 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 effectively. 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 at UCLA's Plasma Physics Group have been developing and using a solution to meet those design criteria. Our solution is based on the Macintosh Operating System using PowerPC-based Macintosh (Power Mac) hardware; we call it a Mac cluster. [2] We use the Message-Passing Interface (MPI) [3], a dominant industry standard [4]. In our ongoing effort to improve the 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, [5] we are seeing the convergence of the best of Unix with the best of the Mac.

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

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 84 – 90, 2005.

user can operate it. We then describe technical details 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.

## 2 The User's Experience Operating the Cluster

### 2.1 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:

**Collect the Hardware.** Power Mac G4s or G5s, one Category 5 Ethernet cable with RJ-45 jacks per Mac, and an Ethernet switch. For each Mac, plug one end of a cable into the Ethernet jack on the Mac and the other end to a port on the switch.

**Configure the Machines.** Making sure each Mac has an working Internet or IP connection and a unique name, specified in the Network and Sharing System Preferences.

**Install Software.** To operate the cluster, a version of the Pooch software package is downloadable. [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 found in no other cluster type.

### 2.2 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 application such as AltiVec Fractal Carbon, available for free download. [6] This initial test also trains the user to accomplish the basic tasks required to run a parallel job. We have distilled primary cluster operation into three fundamental steps:

**1. Selecting an Executable.** After the user selectes New Job… from the File menu of Pooch, the user may drag the AltiVec Fractal Carbon demo from the Finder to this Job Window, depicted in Figure 1.

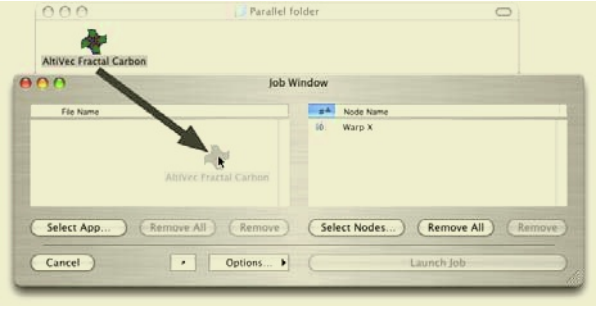


**Fig. 1.** To set up a parallel computing job, the user drags a parallel application, here the Fractal program, and drops it in the Job Window of Pooch

**2. Selecting Computational Resources.** Next, the user chooses nodes to run in parallel by clicking on Select Nodes…, which invokes a Network Scan Window, shown in Figure 2. Double-clicking on a node moves it to the node list of the Job Window.

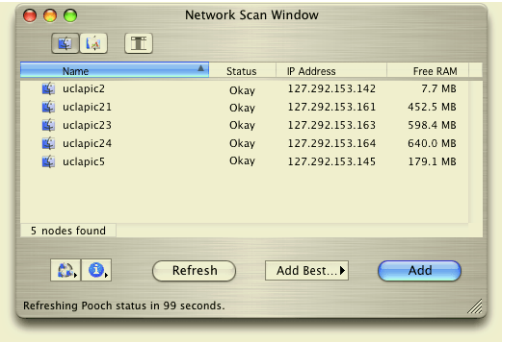


**Fig. 2.** Selecting nodes is performed using the Network Scan Window, invoked by clicking on "Select Nodes…" from the Job window

**3. Combining These Selections Through Job Initiation.** 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 demo then calculates its achieved performance, which should be significantly greater than single-node performance.

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. We chose to use a graphical user interface (GUI) because a GUI tolerates the type of error and imprecision that users

can accidentally introduce when operating any device. This use of a GUI is meant to contribute to the efficiency with which the user can operate the cluster.

### 2.3 Debugging on a Mac Cluster

So that the Plasma group's physics researchers can maximize their time studying physics, we 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.

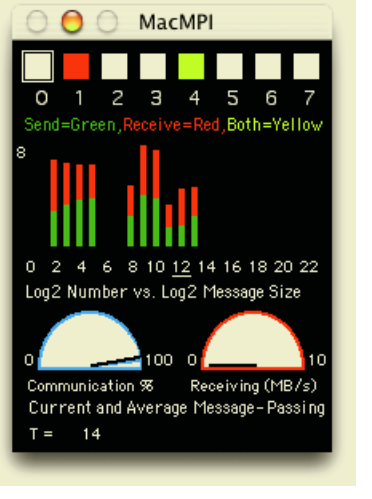


**Fig. 3.** The monitor window of MacMPI, which keeps track of statistics about the execution of the running parallel application

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, the moment 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.

# 3 Design Implementation

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.

## 3.1 MacMPI

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 forms: 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] We achieve excellent network performance comparable to other implementations.

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.

## 3.2 Pooch Application

Pooch is a parallel computing and cluster management tool designed to provide users accessibility to parallel computing. As of this writing, the latest version was released in September 2004. 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: Carbon, Cocoa, Mach-O, Unix shell scripts, and 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. [6] Because of OS X, MPIs of such varied histories are all now supported in the one environment.

## 3.3 Distinctions from Other Implementations

**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 has work begun on versions that identify distinctions between these tasks, such as the emerging MPICH2 rewrite of mpich. [7]

**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. 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.

**Taking 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 Macintosh operating system. We are taking advantage of Apple's commercial, non-scientific motivation to provide a consistent, reliable, well-behaving API, operating system, and hardware.

**No Static Data.** No assumptions have been made about particular hardware at particular addresses being available. We rely on dynamically determined network information, automatically eliminating a host of potential sources of failure that the user might encounter. 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.

**Minimum 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 configuration requirements are that the node has a working network connection with a unique IP address and a unique network name, requirements already in place for web browsing and file sharing. This design has great implications for the mainstream because end users do not wish to be concerned with configuration details.

**Minimum Centralization.** A common philosophy used to increase the performance of parallel codes is to eliminate bottlenecks. Extending that concept to clustering, we eliminated the "head node" of the typical Linux-based cluster. Linux clusters require shared storage (NFS, AFS, etc.) to operate, yet it is a well-known single point of failure.

We chose a decentralized approach. 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 and flexibly choose how to combine nodes for cluster computation from job to job.

## 4 Conclusion

The inexpensive and powerful cluster of Power Mac G3s, 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.

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 raw processor power) when 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 the "plug-and-play" technology to make that vision come true.

## Acknowledgements

Many people have provided us useful advice over the last few years. 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.

## 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] 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.

[3] 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].

[4] 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

[5] http://www.apple.com/macosx/

[6] See http://daugerresearch.com/pooch/

[7] http://www-unix.mcs.anl.gov/mpi/mpich2/

[8] http://developer.apple.com/documentation/CoreFoundation/Networking-date.html