“Plug-and-Play” Cluster Computing Brings HPC to the Mainstream

Dean E. Dauger - Dauger Research, Inc.

Abstract

To achieve accessible computational power for our research, we developed, on the Macintosh platform, the tools to build easy-to-use numerically-intensive parallel computing clusters. We find that the usability and reliability of the Mac cluster 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. By "reinventing" the cluster computer, we provide a unique solution designed to maximize accessibility for users.

To support numerically-intensive and tightly-coupled problems that demand the computational power and networking capabilities of clusters of Macintoshes, our software technology supports five implementations of the Message-Passing Interface (MPI), today a dominant industry standard. MPI's status implies how its underlying computing paradigm has revealed itself to be the most efficient and economical way yet found to apply large numbers of processing "cores" effectively for general purposes.

We present two new applications of our approach to clustering:

1. The new Supercomputing Engine for Mathematica enables Wolfram Research's *Mathematica* to be combined with the programming paradigm of today's supercomputers. In contrast to typical master-slave "grid"s, this solution instead closely follows MPI, from inside the *Mathematica* environment, and has every kernel in the cluster communicate with each other both directly and collectively, necessary to address the largest problems in scientific computing.

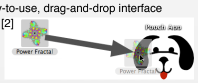
2. Modern compression/decompression (codec) algorithms for video processing increasingly have compression times that easily exceed playback times. We address this emerging computational demand using MPI-based cluster computing. We implement load-balancing parallelization of QuickTime video compression, including frame-reordering H.264, and interface our cluster support software with mainstream desktop video-editing applications such as Final Cut Pro.

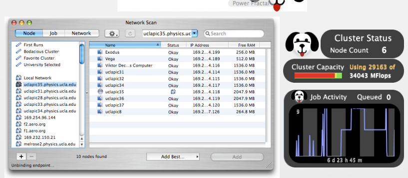
Inventing the Mac Cluster

Plug-and-play cluster computing

Our group was the first to build a Mac cluster, one using Macintosh hardware and operating system. Starting in 1998, we wrote MacMPI, the first Message-Passing Interface implementation for the Macintosh, which enabled parallel codes that ran on the largest supercomputers to run on the Mac. [1]

Since that time, we evolved the technology to use TCP/IP and Mac OS X, even clustering Intel-based and PowerPC-based Macs. The latest incarnation of the user interface is the Pooch Application, featuring the only modern, easy-to-use, drag-and-drop interface to parallel computing. [2]



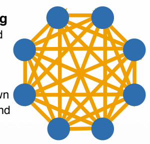


Parallel Paradigms

The reign of the single-processor computer is over

The popularized version of Moore’s law, expecting doubling performance per processor, has come to an end. Finding no other avenue, processor makers instead offer chips with multiple “Core”s. Software writers, in order to best use multiple Core hardware, must choose an efficient programming paradigm. [3]

Distributed-memory message-passing

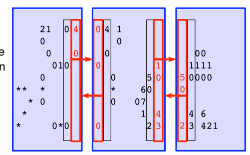


Inherited from 25 years of lessons experienced in the high-performance computing (HPC) industry, the paradigm of distributed memory MPI (Message-Passing Interface) assumes multiple processing cores operate with their own exclusive section of memory and share data and instructions with one another via a network.

Because this parallel computing paradigm covers a wide variety of processing patterns while using hardware economically, this approach is so widely used today in HPC that MPI is the de facto, portable standard at supercomputing centers worldwide.

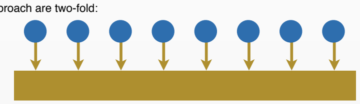
Example: Parallel Life

Conway’s “Game of Life” can be implemented in parallel using an “nearest-neighbor” message-passing pattern. [4] Messages maintain the edges of each partition of the problem space.



What’s wrong with shared memory and threads?

This parallel computing paradigm assumes a series of processing threads that all have shared access to all memory of the system. The problems with this approach are two-fold:



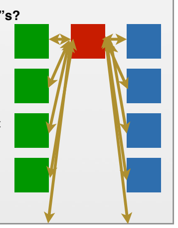
• Software: Because memory is shared, threads may step on each others’ work, potentially giving erroneous results randomly. Determinism, formerly a defining feature of a computer, is easily obliterated, requiring the programmer to track down and eliminate such non-determinism. E. A. Lee of UC Berkeley writes in *The Problem with Threads*: “… we in fact require that programmers of multithreaded systems be insane.” [5]

• Hardware: Data is commonly served from memory using a shared bus that easily can be overwhelmed by the transaction requests of the processing cores. Beyond 16 cores, this memory bus is so taxed that hardware makers must design much more expensive, complex technology to compensate.

Silicon Graphics, Inc., exclusively championed this approach in HPC until declaring bankruptcy in 2006. The same approach is in practice in all currently shipping Intel-based Macs and PCs.

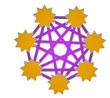
What’s wrong with master-slave “grid”s?

Common “grid” implementations have one manager or “controller” machine that collects work from source applications or “clients” and forwards them to computing nodes or “agents”. HPC practitioners easily recognize the bottlenecks inherent in this “master-slave” scheme. While it works for simple problems that have negligible communication needs, HPC would never consider such an approach for general-purpose parallel computing due to data congestion issues inherent in this design. [6]



Supercompute *Mathematica*

Supercomputing Engine for Mathematica

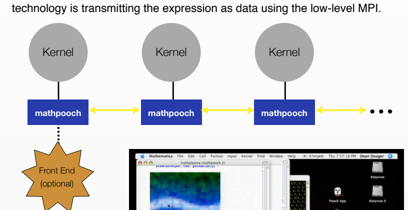


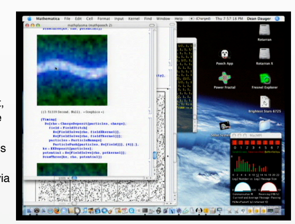
We began by implementing an MPI library within the *Mathematica* environment, an industry first. Combining our easy-to-use, patented Pooch cluster technology with Wolfram Research’s *Mathematica* creates a technology with unprecedented capabilities neither could do alone.

In addition to implementing and extending a series of MPI calls to *Mathematica*, high-level communication and processing calls that implement common communication patterns based on our experience with parallel codes. The result is a merge of the latest ideas from HPC and *Mathematica*. [7]

Structure

Applying the paradigm of distributed-memory MPI to *Mathematica*, our technology launches multiple instances of the *Mathematica* kernel, each under the control of an instance of our mathpooch module. These mathpooch modules constructs and uses a low-level MPI network communications layer. Expressions transmitted from any kernel are intercepted by mathpooch, forwarded between mathpooch’s using the low-level MPI, then recreated in the target kernel elsewhere on the cluster. For the *Mathematica* environment, this process creates the illusion that *Mathematica* is calling MPI, but in fact our technology is transmitting the expression as data using the low-level MPI.





The *Mathematica* Front End, if present, connects to instance #0 of mathpooch. It intercepts commands from the Front End and forwards them via the mathpooch’s on the cluster to all the kernels. The kernels then perform their work and coordinate using MPI like modern supercomputers, and the Front End can display the results.

The calls

The API of the Supercomputing Engine is divided into three categories:

• Low-level MPI: Point-to-point transmissions, synchronous and asynchronous
• Collective MPI: Communications involving any subset of processors
• High-level communcations: Implement commonly used tasks, behaviors, and communications patterns present across parallel computing.

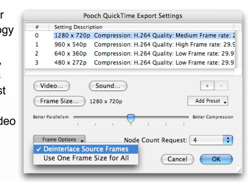
Since “Everything is an Expression” in Mathematica, subroutines, functions, graphics, sound, and equations can be sent via MPI, not just data.

SEM Library	Supported Calls
Low-level MPI	mpiSend, mpiRecv, mpiIsend, mpiIRecv, mpiTest…
Collective MPI	mpiBcast, mpiAlltoAll, mpiReduce, mpiCommSplit…
High-level	ParallelTranspose, ParallelNIntegrate, EdgeCell …

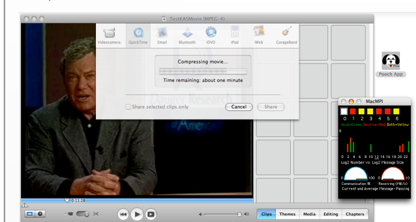
Clustering QuickTime H.264

Pooch QuickTime Exporter

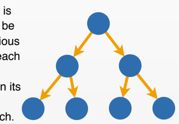
The first to accelerate H.264 encoding using MPI-based clusters, the Pooch QuickTime Exporter plug-in accepts data from any video editing application that uses the QuickTime component architecture and redistributes it onto a Mac cluster for parallelized encoding by any QuickTime-supported codec. [8]



For content creators and other video producers, this technology can produce simultaneously multiple compression formats, varying both in codec settings and frame sizes, to make most efficient use of limited source speed (15 fps for HD) from video editors like Final Cut, iMovie, and After Effects targeted for Apple TV, websites, iPods, DVDs, and other devices.



For video streaming situations, a binary tree communications pattern, supported by MPI, is appropriate because the current frame may be dependent on the data from any of the previous frames. Here it would be possible to have each processor on the cluster compress only its assigned portion of the video feed and return its compressed section as it monitors the feed. This is impractical in a master-slave approach.



References

1. http://daugerresearch.com/vault/aua.shtml
2. http://daugerresearch.com/pooch/recipe.shtml
3. http://daugerresearch.com/vault/parallelparadigm.shtml
4. http://daugerresearch.com/vault/parallellife.shtml
5. http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.html
6. http://daugerresearch.com/vault/parallelzoology.shtml
7. http://daugerresearch.com/pooch/mathematica/
8. http://daugerresearch.com/pooch/quicktime/

Portions of this work are patented or patent-pending. The Supercomputing Engine for Mathematica was produced in partnership with Zvi Tannenbaum of Advanced Cluster Systems plus the assistance of Wolfram Research.

Thanks goes to Viktor K. Decyk, the UCLA Plasma Physics Group, and the Applied Cluster Computing group at NASA’s Jet Propulsion Laboratory for their support over the years. Particularly, our thanks goes to the late John M. Dawson for his support at the most sensitive stages of this work.