

Applying High-Performance Computing Technologies to Industrial Manufacturing

Dave Hardin, PE, CSDP
Consulting Software Engineer
Invensys Systems Inc.
Foxboro, MA 02035

KEYWORDS

Industrial manufacturing, high-performance computing, computational cluster, MPI, Message Passing Interface, Beowulf, parallel computing

ABSTRACT

High-performance computing grew from the basic need to solve very large, complex and computationally intensive problems in the minimum time possible. These computing platforms, called supercomputers, consisted of a large number of close-coupled processors wired together with a high-speed interconnect running under a proprietary operating system. From an application point-of-view, the fundamental concept was one of breaking up a large problem into smaller “chunks,” executing each “chunk” on its own processor and reassembling the individual results upon completion. This gave rise to what is commonly referred to as “parallel computing.” Parallel technology has traditionally been relegated to the back-offices of government laboratories and research universities due to its high financial and intellectual cost. Over the past decade, these close-coupled, special-purpose supercomputers have evolved into commodity, “off the shelf” clusters, also known as “clusters of workstations.”

Commodity clusters basically consist of a set of personal computers wired together using a high-speed network. They come in all sizes and range from several 3GHz dual-Xeons to over 500 1GHz Pentium III's. At the high-end, peak performance of over 300 gigaflops has been achieved, at a significantly lower cost than a comparable supercomputer. Also during this period, an effort arose to develop a standard ‘C’ language API (Application Programming Interface) for developing portable, parallel-aware applications called MPI, the Message Passing Interface. MPI runs on the vast majority of clusters and has become the de facto-standard API for developing message-based, parallel applications. Because of their low cost and ease of implementation, commodity clusters are starting to bring the power of high-performance computing to a far larger audience than ever before. Is there a place for this technology within industrial automation? What opportunities exist within industrial manufacturing that could benefit from this technology? Is the technology solid enough to make its move into manufacturing?

Where and how can it be applied? Once the hardware has been assembled, is the software within reach of domain experts or does it still require special expertise?

This paper will attempt to address these questions while providing a technical overview of the hardware and software systems used to build clusters.

INTRODUCTION

The worlds of academic/scientific high-performance computing (i.e., HPC) and industrial manufacturing have been evolving separately for many years and it appears that they may be on a “virtual” collision course. Before delving into technical issues, let’s take a quick look at these two environments and how they have evolved over the past decade.

High-performance computing has always been about pushing the limits of computational technology in an effort to solve very complex mathematical problems. These problems often involve computing the solution to linear and non-linear models that consist of a very large number of unknowns and equations. The importance of solving these problems has justified the cost of developing systems that contain large numbers of processors that can all be directed toward solving a single problem. Research institutions and the US government have been key factors in moving high-end computing technology forward. The desire to solve “Grand Challenge” problems in the areas of weather forecasting, high-energy physics and cosmology all the way to protecting the nation’s nuclear stockpile through the use of full-physics modeling, has driven the required investment. Standard commercial computing systems, while functional, could not provide timely solutions. In fact, it’s all about time. The value of information decays with time. Answers are of value if there is still time to react upon the answer and thus affect the outcome. In essence, what is the value of forecasting tomorrow’s weather if the model’s solution takes three days to compute? Time is the valuable commodity. One could characterize these systems as being equivalent to highly-tuned, temperamental race cars, both needing a lot of specialized “care and feeding.” The cost of this high maintenance was accepted as long as performance was delivered. The old adage “Speed costs money, how fast do you want to go?” rings true.

On the commercial side of the street, industrial manufacturing has typically been driven by bottom-line economics and heads-up competition with the need to produce more widgets, or refine a barrel of oil, at an ever-decreasing cost. Capital investment in manufacturing facilities undergoes rigorous ROI (Return On Investment) analysis. If one looks at downstream gasoline refining and marketing, a technology investment at a refinery, such as a DCS modernization or advanced application, is balanced against the same dollars invested in constructing new gas stations. Technology investments in the manufacturing environment must bring with them strong value propositions and pay for themselves with very tangible benefits. Another facet of manufacturing is that computer systems are very often considered “embedded” and essentially part of the furniture. They are expected to run continuously and provide uninterrupted, “24x7” service. Operations management frowns upon failures and upgrades that result in the perturbation of normal operations. Systems that are complex and require highly skilled talent to maintain must produce results in order to remain viable. These systems can best be compared to the family sedan or SUV. While a few horses under the hood can help differentiate models, it is system cost, reliability and support that really matter. The trend toward outsourcing information technology and

system support further illustrates the desire to reduce system infrastructure expenses. Driven by economies of scale and the maturing of technology, the PC has evolved into the general-purpose application execution platform of choice in manufacturing. Balancing this drive toward simplicity is the fact that manufacturing processes are complex, heavily instrumented and highly automated.

Manufacturing process control and information systems contain large numbers of processors, each focused on addressing a specific problem. These systems generate vast quantities of data on a real-time basis as feedstock is processed and products are produced. This process data must be stored, filtered, aggregated, modeled, reported, trended, disseminated and ultimately acted upon by those responsible for the profitable operation of the facility. (Note: Most process data ends life squirreled away in a data grave called the process archival system.) The production manufacturing cycle essentially consists of the following phases: long-term planning, short-term scheduling, real-time execution, and as-produced accounting. Each cycle builds on the one that came before as information is accumulated and understood. Comparing the planned versus actual production, in context, allows for continuous improvement of the manufacturing cycle and thus more efficient and profitable operation. (In control speak, one could think of this cycle as being a large, complex, feed-forward control loop with feedback trim.) Systems and people that provide ancillary, but required, services such as environmental monitoring, safety, maintenance, plant engineering, quality control and process engineering, support the production process.

An important characteristic of these production phases is how they “view” time. The planning and scheduling systems peek into the future. The execution systems run in the present and the accounting and auditing processes look at past performance. This leads to the requirements that; 1) production variables provide comprehensive coverage and are accurately measured, 2) control execution is stable, robust and occurs in real-time and 3) planning and scheduling is based on comprehensive models that can accurately predict future production. This is seldom the case in real life. Approximations and simplifications abound. A subset of the optimum required process variables are often measured. Fudge factors are introduced and new constants are created with best guess values. Control systems are often designed with a bias toward repeatability over absolute accuracy. Planning models, advanced control models and optimization models are by nature complex and contain simplifications in an effort to achieve “good enough” accuracy while converging on a solution in a reasonable amount of time. Material and heat balances follow suit. There’s a lot of room for improvement in most manufacturing operations.

In an effort to compensate for this indigenous complexity, many manufacturers attempt to achieve, and maintain for as long as possible, steady state operation. This implies keeping the input and the output constant. On the supply side, this is accomplished through the use of long-term contracts for feedstock with known quality. An interesting scenario arises when one deals with feedstock on the spot-market where values abound, but quality and quantity can vary widely. An accurate and timely, multi-period solution to the manufacturer’s planning LP (i.e., Linear Program) could be extremely valuable in the bidding decision process. On the output side, keeping the product slate constant does not allow the flexibility to pursue ad hoc opportunities that arise in the market. The alternative to steady state operation is to move toward flexible, dynamic operation. This implies high performance, high-fidelity process, process control, process optimization and planning models, data analysis and data mining tools for model validation and discovery, and equipment failure prediction. As the time horizon of economic process optimization moves from weeks and months to days and hours, the need for timely, accurate

identification of effective product and component value increases. These values are quite often implicit and must be calculated based on inferential techniques which can be compute and data intensive. What information patterns and relationships could unsupervised neural agents discover? Could these agents uncover subtle but important changes in process dynamics that might have an adverse affect on the models? Could these agents detect non-modeled, off-normal conditions? Could an accidental environmental emission be predicted and quantified prior to actual physical detection and impact? Could these agents detect the pending mechanical failure of critical rotating machinery? All of these and many more are candidates for high performance technology.

During the past decade, hardware and software technologies have undergone very rapid expansion, fueled in part by the Internet feeding frenzy of the late 1990's. The financial resources that were invested during this period in systems technology and network infrastructure are experienced daily around the globe as people become more and more dependent upon the Internet and the information that it contains. Thanks to Moore's Law (i.e. circuit density doubles approximately every 18 months), the networking technology available as commodity product has eclipsed the gigabit per second barrier and CPU speeds have reached several gigahertz. One terabyte of storage costs about \$1000 with RAID support, a standard option on motherboards. Dual SMP (Symmetric Multiprocessor) systems can be purchased for a small premium above the cost of an additional CPU. One 19" equipment rack can hold 32 of these processors in a form factor that requires about 6 square feet of real estate, packing an astounding 100 GHz of loosely-coupled compute capacity. This technology represents today's value/performance sweet spot, with blade servers promising even greater packing densities. Above this level, as systems move upscale toward 8, 16 and 32 processors in a close-coupled, scale-up, SMP architecture, the cost increases exponentially. This is due to the need for proprietary technology and the reduced volumes over which development funding can be amortized.

The industrial, academic and government communities have all taken notice of this trend. The question quickly became "How best can this commodity technology be applied to solve problems that were previously relegated to high-priced, specialized systems and solutions?" Academia rose to the challenge and wired a large number of PC's together into what is now called a scale-out, "Beowulf" cluster. The science and research laboratories followed suit and scaled Beowulf into world-class supercomputers such as the MHPCC's (Maui High Performance Computing Center) Huinalu Linux SuperCluster, which contains 520 x 1GHz Pentium III's with a theoretical peak performance in the area of 485GFlops(1). During this time, cluster technology was applied to areas such as financial modeling, computation fluid dynamics, human genome bioinformatics, protein folding, data mining and seismic analysis. Cookbooks were even published to help assist in constructing Beowulf Clusters. (2)(3)(4)

As a quick review:

- High Performance Computing deals with focusing the work of many processors toward solving a single problem.
- Manufacturing typically focuses the work of many processors toward solving orthogonal problems.
- Beowulf clusters are transforming the world of high performance computing.
- Manufacturing could use more compute and data analysis horsepower.
- Hardware has become very inexpensive and easy to assemble into piles of PC's interconnected by Gigabit Ethernet.

The Hardware

A typical, rack-mounted cluster might contain the following:

- 1 – 19” rack
- 16 – Rackmount Servers, Dual Xeon 2U 2.4 GHz w/ 1 GB memory, 240 GB SATA disk (\$1850/ea)(5)
- 1 – Memory Expansion
- 1 - Gigabit Ethernet Switch

(Note: If the application has a high degree of internode coupling, several low-latency networking options exist at extra cost, such as the Myricom Myrinet fiber network. (6))

The hardware for the above system could be assembled for an investment of \$40-\$60K, with essentially zero risk. If desired, the cluster could be morphed quickly into a set of standard, high-density back office servers. This is well within the financial resources of many companies. Inside of a manufacturing site, this rack would be very much at home in a computer room sitting next to the email, file and web servers. In fact, the same organization and system management toolset that supports the existing back office infrastructure could build and support the new cluster with minimal incremental cost. Our system is looking good, but...

As Greg Pfister so aptly wrote almost ten years ago, “The problem is not hardware. It’s software.” (7) The branches of the software decision tree include the operating system and system management tools, parallel middleware, and the application software itself. Many viable options exist and the following discussion does not attempt to be comprehensive.

The Operating System

The two primary options for commodity cluster operating system software are (surprise!): Linux and Windows Server 2003. Linux is the most widespread and popular choice while Windows is the new kid on the block. If the manufacturing IT shop has strong UNIX capability, then Linux may be the obvious choice. If the IT shop is Microsoft, then Windows Server 2003 might be the logical choice. The dominance of Microsoft Windows on the corporate desktop along with the growth of the Microsoft back office environment has resulted in Windows being a viable contender in the cluster computing space. To further support this, Microsoft published a white paper that specifically targets HPC applications called “Solution Guide for Migrating High Performance Computing (HPC) Applications from UNIX to Windows, Version 1.0.” (8) A sampling of software products for Windows Clusters is also available on the “Computational Clustering Technical Preview Toolkit” CD available from Microsoft. (9)

Both environments have a rich set of system development and support tools, and both can perform quite well.

The Task Scheduler

Clusters can be classified as shared or dedicated. Shared clusters support many users and should utilize a batch queuing system, reminiscent of the IBM JCL batch processing days of old. Batch queuing requirements for a dedicated cluster are application dependent. If the application executes periodically and requires all the resources of the cluster, serial, self-scheduled execution may be adequate. This would apply to the execution of online models and optimizers as well as real-time pattern discovery and change detection engines.

Two open source task schedulers in wide use are Condor and the Maui Scheduler. Condor provides a queuing and matchmaking service that mates the requirements of an application with the resources available in the cluster. (10) The Maui Scheduler is a job scheduler with many advanced features for Linux clusters. (11)

Commercial schedulers include the ClusterController from MPI Software Technology. (12)

Next we need network messaging middleware that can tie together all the processors into one coherent system.

The Parallel Glueware

In order to build application software for a cluster, it's necessary to use message-passing middleware to abstract the cluster and to facilitate application development. Two main flavors are available: MPI (Messaging Passing Interface) and PVM (Parallel Virtual Machine).

MPI is very powerful, well supported and runs on many Linux and Windows clusters. It was developed in 1995 as an abstraction layer for parallel computing in order to facilitate application portability between hardware platforms. This effort has proven very successful with MPI becoming the defacto standard messaging software used in high performance computing. The open source reference port of MPI is called MPI-CH and is available for download from the Argonne Labs website. (13) Several commercial versions of MPI are also available from vendors such as MPI Software Technology. (12) These provide additional value such as technical support and support for a wider range of networking options.

MPI provides an API (Application Programming Interface) with 'C' and FORTRAN language binding that provides a message-based model for inter-process communications that is independent of the network transport and physical communication layers. (14) Within an MPI cluster, processes are grouped into "communicators" and each process has a unique "rank" identifier within the communicator. The rank of '0' is special and can be considered the overall manager process. Using this rank, each process can determine its role and resources within the overall communicator. In addition, the communicator and rank is used as the basis for addressing and sending messages to all other processes in the cluster. This is where MPI earns its keep. The following is a very, abbreviated summary of the messaging capability available: (15)

- Point-to-Point Messaging - Between processes
 - Blocking Reads/Write – Wait for completion
 - Non-blocking Reads/Writes – Return immediately, don't wait
- Collective Messaging – Within a communicator
 - Synchronization
 - Barrier – Wait till all other processes get here
 - Data Movement
 - Scatter – Send message to all other processes
 - Gather – Get message from all other processes
 - AllGather – Gathers values from all processes and distributes to all processes
 - Broadcast – Send message from Rank 0 to all other processes
 - Computations
 - Reduce – Applies an operation to all processes and sends result to one process

In order to promote portability between machines with differing architectures, MPI defines a set of standard data types from which messages are constructed.

Advanced “virtual topologies” are also available that enable an application to map into a Cartesian coordinate system and chat with other nodes in a straight forward manner. This adapts well to solving “wire-mesh” types of applications.

PVM uses another API abstraction for message passing that also hides the details associated with the underlying communications machinery. (16)(17) An important feature of PVM is the ability to spawn dynamic processes and process groups. This allows for additional runtime flexibility and adaptation. MPI does not support dynamic processes under Version 1, but does under Version 2.

Now we need the most important ingredient, application software. This is where the fun begins...

The Application Software

Building application software for parallel clusters is not easy. It has traditionally required advanced software talent. Can the process of building parallel applications be simplified to the point where domain experts and corporate software engineers can handle the job? The answer is yes and no. If the application has a high degree of node coupling, then the software development effort can be significant. Tight coupling implies that a node must exchange data with other nodes during the course of execution. On the other hand, loose coupling implies that each node can work independently to achieve a solution. A master node then assembles and displays the results upon completion. A lot of problems fall into the category of being loose-coupled and are often called “embarrassingly parallel” applications. These are the applications that fall naturally into a commodity cluster. The high network traffic of very close-coupled applications still requires the ultra-low latencies of a parallel Supercomputer. Bottom line, not all applications are suited for execution on a commodity cluster. In an effort to “crawl/walk/run,” we will assume for the present that cluster applications fall into the realm of loose-coupled although MPI

has a class of message passing functions that specifically address a grid topology and the ability to exchange data with a node's closest neighbors.

How can one determine the speedup of an application running on a cluster? The compute time must be significantly greater than the data transfer time over the network for an application to scale well on a cluster. Amdahl's law describes this application speedup and scalability in detail.(18) In an ideal situation, a 16-processor system should achieve a 16 to 1 performance boost. In real life, this cluster may achieve a 10 to 1 speedup, thus reducing an applications execution time from a baseline of 60 min. to 6 min. While less than ideal, this is still a very sizable increase in performance.

Probably the best-known solution pattern for clusters is called the "Master/Worker" pattern. This pattern uses one node as a master node and all the other available nodes as worker nodes. The "Master/Worker" mirrors standard organizational behavior of work groups. The boss divvies out work assignments to his/her team members and waits for the work completion emails to roll in. As soon as a member completes an assignment, another task is assigned, thus preventing idle web surfing. This continues until all the work assignments are complete and the boss declares victory with a team party. MPI and PVM both map into this scenario quite well. In addition, this pattern can be adapted quickly to solving many flavors of applications, not just those that are computation-intensive. The work performed can also be data-intensive, I/O-intensive or some combination of the three. The important feature that distinguishes a cluster application from other normal applications is that all of the processes are working together to solve a single problem. This is the recommended starting point for partitioning the workload and designing the cluster application. Code construction, testing and debugging would follow.

Many more advanced parallel algorithms and patterns have been identified. Ian Foster's book "Designing and Building Parallel Programs" drills into this topic in great detail. (19)

Parallel Math Applications

Parallel applications that are primarily compute-oriented can be constructed with the help of two popular mathematics applications on the market: Mathematica and Mathworks Matlab. Mathematica offers gridMathematica with the Parallel Computing Toolkit and information is available that describes the application of MPI technology to Matlab.(20) (21)(22) Both approaches leverage and extend powerful math solver packages.

.NET and Future Opportunities

Cluster software development is not as easy as it should be and has yet to achieve widespread acceptance. It still hasn't entered the realm of the "Corporate Developers" around the world that build Microsoft Office and Visual Basic applications on a daily basis. These applications are responsible for a significant portion of the reporting, viewing and manipulation of existing production data. Spreadsheet calculations, in particular, form the basis for large numbers of offline models used in manufacturing. Clearly the cost of entry into cluster computing is still too high and needs to be lowered even further.

Enter .NET. The ability to construct a parallel application using the .NET languages and development tools is now possible, thanks to the work at Cornell and Indiana University. (23)(24) A team at Indiana University created MPI.NET that consists of a set of wrappers over the MPI 'C' API. This allows .NET apps to directly access the messaging capability of an MPI-based cluster. The ability to use Visual Studio.NET, C# and Visual Basic.NET while still maintaining cost-effective application speedup may help trigger widespread application of cluster technology.

Many opportunities exist for enterprising companies to exploit commodity clusters. These range from building cluster-aware applications to supporting Remote Computational Clusters with consulting services and application development assistance. Remote clusters could provide users with access to high-performance technology without the need for locally installed and maintained hardware.

Conclusion

Commodity Cluster technology has matured quite a bit over the last decade and is capable of achieving cost-effective, 10-30 gigaflop performance. The hardware is ready for prime time but the software still needs the touch of a knowledgeable software engineer. In the hands of the technically astute, clusters can become very valuable weapons in the war to help manufacturers achieve a competitive advantage in the market place.

References

- (1) <http://www.mhpcc.edu/doc/huinalu.html>
- (2) Sterling, Thomas, Salmon, John, Becker, Donald J., Savarese, Daniel F., "How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters (Scientific and Engineering Computation)", MIT Press, Cambridge, MA, 1999
- (3) Sterling, Thomas, "Beowulf Cluster Computing with Windows" (Scientific and Engineering Computation)", MIT Press, Cambridge, MA, 2001
- (4) Sterling, Thomas, "Beowulf Cluster Computing with Linux" (Scientific and Engineering Computation)", MIT Press, Cambridge, MA, 2001
- (5) <http://www.tigerdirect.com>
- (6) <http://www.myri.com/>
- (7) Pfister, Gregory, "Why Clusters?", "In Search Of Clusters", Prentice Hall PTR, Upper Saddle River, New Jersey, 1995

- (8) White Paper, "Solution Guide for Migrating High Performance Computing (HPC) Applications from UNIX to Windows Version 1.0", Microsoft, <http://www.microsoft.com/windowsserver2003/hpc/default.mspx>
- (9) <http://www.microsoft.com/windows2000/hpc/toolkit.asp>
- (10) <http://www.cs.wisc.edu/condor/>
- (11) <http://www.supercluster.org/maui/>
- (12) <http://www.mpi-softtech.com/>
- (13) <http://www-unix.mcs.anl.gov/mpi/mpich>
- (14) <http://www-unix.mcs.anl.gov/mpi/standard.html>
- (15) <http://www.mhpcc.edu/training/workshop/mpi/MAIN.html>
- (16) Geist, Al, Beguelin, Adam, Dongarra, Jack, Jiang, Weicheng, Manchek, Robert, Sunderam, Vaidy, "PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing", MIT Press, Cambridge, MA, 1994
- (17) http://www.csm.ornl.gov/pvm/pvm_home.html
- (18) Morse, H. Stephen, "Can Parallel Machines be used Efficiently?", "Practical Parallel Computing", AP Professional, Boston, MA, 1994
- (19) Foster, Ian T., "Designing and Building Parallel Programs", Addison-Wesley Publishing, Reading, MA, 1995
- (20) <http://www.mathworks.com/>
- (21) <http://www.wolfram.com/>
- (22) <http://www.wolfram.com/products/gridmathematica/whatis.html>
- (23) Paper, Lifka, David, Walle, Lucia, Zaloj, Veaceslay, Zollwed, John, "Increasing the Accessibility of Parallel Processing with Microsoft .NET", Cornell Theory Center, <http://www.ctc-hpc.com/Papers/PPD2002.NET.pdf>
- (24) Willcok, Jeremiah, Lumsdaine, Andrew, Robison, Arch, "Using MPI with C# and the Common Language Infrastructure", Indiana Univ., IN, <http://www.cs.indiana.edu/pub/techreports/TR570.pdf>