IMPLEMENTING A PARALLEL COMPUTING LABORATORY FOR UNDERGRADUATE TEACHING AND RESEARCH

Michael Fontenot, Kendrick Aung

Department of Mechanical Engineering
Lamar University, Beaumont, Texas 77710

Abstract

Rapid advances in the computer technology and widespread availability of computers have made it possible for many engineering schools to incorporate high performance computing laboratories for undergraduate teaching and research. Many employers now require undergraduate engineering training to include hands-on experience with modern engineering software such as 3-D solid modeling and finite element analysis. Many core and elective courses in the mechanical engineering curriculum require use of engineering software such as AutoDesk Inventor, Working Model, Pro/Engineer, Pro/Mechanica, Nastran, and CFX. In order to meet the increasing demands of computing power, the department has decided to implement a parallel computing laboratory to be used for undergraduate teaching and research activities. This paper describes the development and implementation of a parallel computing laboratory in the Department of Mechanical Engineering at Lamar University. The paper presents implementation of the laboratory including detailed descriptions on hardware, software, networking, testing, and benchmarking.

Introduction

Rapid advances in the computer technology and widespread availability of computers have made it possible for many engineering schools to incorporate high performance computing laboratories for undergraduate teaching and research. Many employers now require undergraduate engineering training to include hands-on experience with computational tools and software packages dealing with 3-D solid modeling, finite element analysis, and fluid flows[1, 2]. To address this issue, the students in the Department of Mechanical Engineering at Lamar University need to use engineering software packages for many of their courses. For example, students use Pro/Engineer and Pro/Mechanica software programs in the mechanical design classes and computational fluid dynamics (CFD) software, CFX, in the computational fluid dynamic course[3]. In order to meet the increasing demands of computing power for course work and research, the department has decided to implement a parallel computing laboratory to be used for undergraduate teaching and research activities. The paper presents design and implementation of the laboratory including detailed descriptions on hardware, software, networking, testing, and benchmarking.

Parallel Computing

Higher processing speed and large memory are essential for investigation and analysis of complex physical problems such as prediction of weather, design of a spacecraft, etc. Parallel computing provides an affordable and cost effective means of developing high performance computing platforms using commodity workstations and PCs. Parallel computing is simultaneous computations of a problem on multiple processors. Most parallel computing can be achieved by using either massively parallel processors (MPP) or a cluster of commodity machines. A cluster is a group of independent computers and thus forms a loosely-coupled multiprocessor system[4]. Each machine in a cluster is referred to as a node. Each machine may have its own memory or memory is shared among many nodes or machines. A network is used to provide communications between machines. The

computation on a cluster is done by developing a parallel program that distributes data and processing to multiple nodes. The most common way to distribute the program and data is by message passing. There are two common ways for controlling and managing parallel processing among many machines: Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). Parallel Virtual Machine (PVM) is a subroutine library developed at Oak Ridge National Laboratory that allows a programmer to create and access a concurrent computing system made from networks of loosely coupled processing elements[5]. PVM allows the cluster to be composed of different platforms, hardware, architecture, and processing power to be networked together and work as parallel machines. Message Passing Interface (MPI) is a standard Application Programming Interface (API) that can be used to create parallel applications. Since PVM is built around the concept of a virtual machine, it has the advantage over MPI when the application is going to run over a networked collection of machines particularly if the machines are heterogeneous. However, MPI has the advantage if the application is going to be developed and executed on a single MPP[6].

Implementation of the Parallel Computing Laboratory

The main objectives of this project are to design and develop a small cluster for parallel computing, and to develop a parallel program to test and benchmark the cluster computing ability. The main goal in developing the lab is to achieve high computing power with a limited budget. The most cost effective way of achieving this goal is to develop the cluster with commodity PCs and gradually increase the number of nodes in the cluster over time. One of the key metrics based on IEEE for measuring cluster performance is $/GFLOPs, where a GFLOPs is defined as 1E9 floating point operations per second. For example, at the 2000 High Performance Networking and Computing Conference, the Gordon Bell prize for the highest price/performance went to a group achieving $920/GFLOPS[7]. As time progresses, due to Moore's Law, now (in 2004) even better price/performance ratios are possible. In addition, utilizing existing or donated hardware can keep the out-of-pocket cost down further. According to the Netlib Repository at Oak Ridge National Laboratory (http://performance.netlib.org), a single Intel Pentium 4 at 2.53 GHz can achieve approximately 1 to 5 GFLOPs depending upon certain criteria. However, other factors such as bus speed and network connection type (i.e., hubs vs. switches vs. fiber) do play an important role in cluster hardware configuration. In the present project, no evaluations regarding network connections and speed have been considered as the main purpose is to gradually build a coarse-grained cluster and increase its capabilities in order to gain more hands-on experience with the system.

Hardware and Software

The computers were obtained from the IBEXPC company (www.ibexpc.com) with the following specifications:

Processor:	Single AMD Athlon 2.4 GHz
Memory:	521 MB DDR RAM
Hard Drive:	40 GB
CD-ROM:	CD-RW drive
Video:	32 MB AGP
OS:	RedHat Linux 9.0

There are many problems commonly encountered in developing a parallel application on a cluster. The main issues are load balancing, bad communications (starvation and deadlock), scheduling of data and process distribution and sequential bottleneck. Therefore, it is necessary to test on a small cluster before implementing the laboratory that will have 16 nodes. Thus, a cluster of 4 machines was used as a test system. Each machine costs about $400 for the specifications given above. The machines are networked using a 10/100 Ethernet hub and CAT 5 cabling. The total cost of the test system is about $1800. One important thing to note is that there is no linear scale up with 100% efficiency as the machines are networked but a

small cluster is an excellent and economical way to start experimenting with clusters.

The operating system chosen for the cluster must be inexpensive, robust, and geared towards networking. Red Hat Linux 9.0 was chosen because of its long history and successes in implementing and operating PC clusters. In addition, Red Hat Linux distribution comes with a large amount of software including GNU compilers for C and Fortran languages and message passing libraries, PVM (Parallel Virtual Machine) and MPI (Message Passing Interface), for building parallel applications.

It is also necessary to keep the learning curve as flat as possible since there are already many aspects to deal with when implementing the cluster. As always, there are tradeoffs involved. It is possible to learn a new data-parallel language and avoid problems such as synchronization but the current trend seems to be to use explicit message passing libraries to maximize generality while still having a basic language (C or FORTRAN) to build applications from. As mentioned before, two most popular libraries for message passing are PVM and MPI. For the present implementation, PVM (http://www.csm.ornl.gov/pvm/pvm_ home.html) was chosen because of its ability to handle heterogeneous computers in a very general way. Similar implementations can be found in other universities and laboratories. For example, the University of Kentucky is using Linux/Fortran/MPI for CFD with turbulence (large eddy) simulations on clusters of PCs with (16) 2.4 Ghz Pentium 4 nodes (Kentucky Fluid Cluster 3).

Cluster Configuration and Initial Programming

Once the machines were obtained, the network needs to be setup and PVM needs to be configured. Only a few publications can be found to the authors' knowledge on parallel computing using PVM[5, 8-10]. Some helpful documents available on the Internet are the tutorial from California Institute of Technology's Center for Advanced Computing Research website (http://www.cacr.caltech.edu) and SCL Cluster Cookbook (www.scl.ameslab.gov). Basic network configuration for a cluster requires designating one machine to be a server and the others as clients. Then, Network File System (NFS) was used to create directories that were mirrored on all machines. Authorization for remote accessing needs to be enabled so that remote login, remote shell, and remote file copying are allowed. At present, the cluster is not connected to the university network as it is still in beta testing phase.

In order to use PVM, a remote hosts file needs to be created and a script file which handles compiling needs to be modified for the particular hardware architecture and directory structure. Ganglia is the software tool used for this project to manage the nodes in the cluster. It is a scalable distributed monitoring system for high-performance computing systems such as clusters and grids. It is based on a hierarchical design targeted at federations of clusters. It relies on a multicast-based listen/announce protocol to monitor state within clusters and uses a tree of point-to-point connection amongst representative cluster nodes to federate clusters and aggregate their state. It leverages widely used technologies such as XML for data representation, XDR for compact, portable data transport, and RRDtool for data storage and visualization. It uses carefully engineered data structures and algorithms to achieve very low per-node overheads and high concurrency. The implementation is robust, has been ported to an extensive set of operating systems and processor architectures, and is currently in use on over 500 clusters around the world. It has been used to link clusters across university campuses and around the world and can scale to handle clusters with 2000 nodes[11].

Once PVM is up and running, it is time to develop a parallel program for an engineering application for benchmarking and testing purposes. The first step in developing a parallel code was to run some examples given in the PVM User's Guide. No problems were

encountered in running the example codes. Thus, a series of code was written that did nothing but pass and alter messages back and forth and write output to files. For example, the first just spawned new tasks. The second printed messages to independent files while altering variables specific to its portion. Since the test cluster is coarse-grained, a parent-child paradigm similar to MPI, instead of a master-slave paradigm where one of the nodes did nothing other than organize communication, was used.

Instead of attempting to write a full-scale parallel code, a simple serial code was written to test out the system. Then, the parallel counterpart of the simple code was written and tested. The plan is to increase code complexity and testing, as well as increase the number of nodes in the cluster in an iterative manner, as more hands-on experience with the system has been gained. The goal is then to calculate speedups (single processor compute time /cluster compute time) and refine hardware/software as needed. This phase has been completed without any problems for executing and completing both serial and parallel codes.

For benchmarking, a one-dimensional transient diffusion algorithm[12] was chosen as a test problem for which there are numerical examples to benchmark against. The test program was written in C language utilizing the algorithmic framework (the SIMPLE finite volume method) and language suitable for the ultimate goal of developing a multidimensional combustion code. An example using a finite-difference transient diffusion problem given in the PVM User's Guide was used as a model for the test code. Writing the serial code was extremely straightforward and quick. Gauss-Siedel scheme was used as the iterative equation solver for the test program. It was decided to use Gauss-Siedel with block (row strip) partitioning for the equation solver. This step turned out to be problematic because the message calls back and forth had to be organized and timed carefully to prevent deadlock. (Deadlock is a state where the program will not terminate because some nodes are waiting for messages from other nodes.) It took some time to resolve the deadlock issue. A search on the Internet found that this was not uncommon, by far. As a matter of fact, papers have been written and parallel programming courses teach sections on deadlock and synchronization. It should be noted that debugging a parallel program was much more difficult than debugging a serial program. Therefore, it is very wise to employ an incremental, iterative approach. It was noted that in many other applications, the issue of deadlock never arose because the program was trivial or the message passing portions were trivial. In general, however, this is not the case. For explicit message passing, the programmer has total responsibility for preventing deadlock. The tentative solution for the deadlock problem was to create a crude barrier using sleep-type functions. Improvements can be made on the current scheme by exploring the PVM barrier function or by creating more sophisticated barriers.

Future Plans

In one semester, a cluster of 4 nodes have been acquired and networked. PVM was installed successfully and a simple serial and parallel code was written and tested gaining hands-on experience in parallel code development. Future plans are to improve the barrier solution and then proceed to expand the code to include convection and multi-dimensionality. The ultimate goal is to develop a parallel application program to simulate the flow and combustion inside an industrial low NO_x burner. Physical models for chemistry and turbulence will be added to the one-dimensional transient code for the diffusion problem. In addition, there are plans to incorporate multi-grid and TVD features to improve numerical robustness. At the same time, benchmark for accuracy and speed on the modified code will be conducted. Once all testings have been done, the cluster will be expanded to 16 nodes. The 16-node cluster will be used for undergraduate teaching and research for the department. For example,

CFX CFD code will be running on the cluster to provide high-performance computing laboratory for students. In addition, the laboratory will be developed further for interdepartmental and interdisciplinary cluster usage.

Conclusions

The price/performance for clusters based on commodity PCs has never been better, continues to improve at an amazing rate. The first costs for entering into parallel computing based on clusters are low and the return on these systems can be extremely beneficial to students as very powerful computing tools are made available to them. Main disadvantages for implementing these systems with limited budgets are lack of standards and technical supports. However, the advantages of building and using such a system from scratch are great and extremely educational in terms of depth of understanding and hands-on learning.

Bibliography

1. Navaz, H. K., Henderson, B. S., and Mukkilmarudhur, R. G., "Bringing Research and New Technology into the Undergraduate Curriculum: A Course in Computational Fluid Dynamics," Proceedings of the 1998 ASEE Annual Meeting & Exposition, 1998.

2. Hailey, C. E., and Spall, R. E., "An Introduction of CFD into the Undergraduate Engineering Program," Proceedings of the 2000 ASEE Annual Meeting & Exposition, 2000

3. Aung, K., "Design and Implementation of Undergraduate Computational Fluid Dynamics (CFD) Course," *Proceedings of the 2003 ASEE Annual Meeting & Exposition*, Nashville, Tennessee, June 2003.

4. SCL Cluster Cookbook, www.scl.ameslab.gov

5. Breshears, C., "A Beginner's Guide to PVM-Parallel Virtual Machine," Joint Institute for Computational Science, Knoxville, Tennessee.

6. Geist, G. A., Kohl, J. A., and Papadopoulos, P. M., "PVM and MPI: a Comparison of Features," Calculateurs Paralleles, Vol. 8 No. 2, 1996.

7. Aberdeen, D., Baxter, J., and Edwards, R., "92cents/MFlops/s, Ultra-Large-Scale Neural-Network Training on a PIII Cluster," Proceedings of the High Performance Networking and Computing Conference (SC2000), ACM Press and IEEE Computer Society Press, November 2000.

8. Dongarra, J., "PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing," MIT Press, 1994.

9. Wilkinson, B., and Allen, C. M., "Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers," Prentice Hall, 1999.

10. Bertsekas, D. P., and Tsitsiklis, J. N., "Parallel and Distributed Computation: Numerical Methods," Athena Scientific, 1997

11. Ganglia, ganglia.source.forge.net

12. Versteeg, H. K., and Malalasekera, W., "An Introduction to Computational Fluid Dynamics: The Finite Volume Method," Prentice Hall, 1995.

Biographical Information

Michael Fontenot is a licensed mechanical engineer and a doctoral student in the Department of Mechanical Engineering at Lamar University. He is working on his dissertation developing a parallel program to simulate the performance of low NO_x industrial burners.

Kendrick Aung is an assistant professor in the Department of Mechanical Engineering at Lamar University. He received his Ph.D. degree in Aerospace Engineering from University of Michigan in 1996. He is an active member of ASEE, ASME, AIAA and Combustion Institute. He has published over 50 technical papers and presented several papers at national and international conferences.