Building Affordable High Performance Computing Platforms for Engineering Education

Yili Tseng
Department of Electronics, Computer, and Information Technology
North Carolina A & T State University
Greensboro, NC 27411, U.S.A.
E-mail: ytseng@ieee.org

YILI TSENG

Yili Tseng received the PhD degree in computer engineering from the University of Central Florida. He is currently an Associate Professor and the advisor of the Computational Technology Concentration in the Department of Electronics, Computer, and Information Technology at North Carolina Agricultural and Technical State University. He published several research papers in internationally recognized journals and conferences, including IEEE Transactions on Parallel and Distributed Systems and Parallel Computing. His research interests include high-performance computing, grid computing, and cloud computing.

Building Affordable High Performance Computing Platforms for Engineering Education

Yili Tseng
Department of Electronics, Computer, and Information Technology
North Carolina A & T State University
Greensboro, NC 27411, U.S.A.
E-mail: ytseng@ieee.org

Abstract - Traditionally, engineering and science disciplines have relied on observation, theory, and experimentation as tools to perform research to explore new knowledge. With the introduction of computer hardware and software, numerical simulation based on mathematical modeling gradually becomes an important tool. After high performance computers are mature and commercially available, numerical simulation has become a tool as important as observation, theory, and experimentation to all engineering and science disciplines. In most cases, it is adopted more often than experimentation because it is more economic, less time-consuming, and able to explore infeasible situations. Without doubts, numerical simulation should be included by engineering education. Numerical simulation depends on high performance computers. High performance computers refer to parallel computers, namely computers equipped with multiple processors. However, the largest barrier to high performance computing education is the high cost of high performance computers. Most institutions cannot afford expensive parallel computers. The author explored and managed to discover three options of affordable platforms suitable for high performance computing classes. The first platform is personal computers equipped with multi-core processors and thread libraries. The second platform is PCs equipped with Graphic Processing Unit cards. The third platform is commodity clusters consisting of inexpensive PCs and network switches. They even can be built with retired PCs. All three approaches provide low-cost solutions for all institutions to offer their high-performance computing education.

1. Introduction

Traditionally, engineering and science disciplines have relied on observation, theory, and experimentation as tools to perform research to explore new knowledge. With the introduction of computer hardware and software, numerical simulation based on mathematical modeling gradually becomes an important tool. [1][2] Most engineering and science disciplines develop numerical modeling for simulations in their respective fields. [1][2][3] After high performance computers are mature and commercially available, numerical simulation has become a tool as important as observation, theory, and experimentation to all engineering and science disciplines. In most cases, it is adopted more often than experimentation because it is more economic, less time-consuming, and able to explore infeasible situations. [1][2] Without doubts, numerical simulation should be included by engineering education. Numerical simulation depends on high performance computers to solve large scale problems or effectively improve accuracy of results because they surpass the computing power of uniprocessor computer system. High performance computers refer to parallel computers, namely computers equipped with multiple processors. [3][19][20][22] Although parallel computers are capable to execute sequential programs, but only one processor is being utilized while other processors sitting idle. Parallel programs have to be written and executed to take advantage of all processors in a high performance computer. Therefore, introductory high performance computing and parallel programming courses should be covered by engineering education as well. Another rationale which makes both courses imperative for engineering education is explained as follows. Since 2003, the speed of uniprocessors can hardly be

pushed because of energy-consumption and heat-dissipation problems. [3] That is why the major processor manufacturers such as Intel and AMD introduce multi-core processors instead of uniprocessors with faster clock cycle since then. Just like multiprocessor computers, parallel programs have to be executed to take advantage of all cores of a multi-core processor. Both phenomena push the need for high performance computing education to be part of engineering education as engineering applications heavily depend on computation.

However, the largest barrier to high performance computing education is the high cost of high performance computers. Most institutions cannot afford expensive parallel computers. Even if a university owns few high performance computers, they are always reserved for research and would not be used for teaching and learning because they are precious. That is the case at the author's institution. In order to find affordable platforms for his high performance computing classes, the author explored and managed to discover three options of affordable platforms suitable for high performance computing classes. The first platform is personal computers (PCs) equipped with multi-core processors and thread libraries. The second platform is PCs equipped with Graphic Processing Unit (GPU) cards. The third platform is commodity clusters consisting of inexpensive PCs and network switches. They even can be built with retired PCs. All three approaches provide low-cost solutions for all institutions to offer their high-performance computing education. They will be presented in this paper so that any institution can select the option which fits their affordability and deploy high performance computing platforms with the minimal costs.

2. Multi-core Processors and Threading Libraries

The easiest way to start parallel processing is to carry out multithreaded programs on multi-core PCs. No extra hardware needs to be added except multithreading libraries needed to be installed. Most multithreading libraries are free. Although multithreaded programs can be executed on uniprocessor PCs, parallelism from multithreading would not make any performance gain because all threads are executed sequentially by a single uniprocessor. Fortunately, multi-core PCs are very affordable these days. The current drawback is that all threads can only be executed on the cores of the same chip. Consequently, only small scale multithreaded programs can be executed because not many cores can be incorporated into a chip by current technology. Nonetheless, considering many-core processors with 32 or 48 cores are being rigorously tested, multithreaded programs will be able to carry out large scale parallel applications soon. It is still worthy to starting teaching multithreading programming. The popular threading libraries and concurrent programming languages are introduced in following subsections.

2.1 Intel Threading Building Block

Intel Threading Building Block (TBB) is the newest threading library which supports C++. [15] It is downloadable at http://threadingbuildingblocks.org for free. It targets to provide high-level parallelism in contrast to low-level parallelism offered by raw threads libraries and Message Passing Interface (MPI). Issues like optimal management of a thread pool, proper distribution of task with load balancing, and cache affinity are automatically taken care of by TBB. Therefore, it is easier for beginning parallel programmers in addition to better performance than the alternatives.

2.2 POSIX Threads

POSIX (Portable Operating System Interface) Threads (Pthreads) is a threading library for C++ as well. It has been existing for more a decade and was the most popular threads library in the past. Hence, more documentations and textbooks are available for Pthreads. [13][14][16]

2.3 Microsoft Windows Threads

Microsoft Windows Threads is a threading library supporting C++ for Microsoft Windows operating systems. It is included in Microsoft Windows SDK. Some programmers prefer it to Pthreads because of the simplicity of use. [24]

2.4 Threads in Java and C# and Tasks in Ada

Java, C#, and Ada are concurrent programming languages which directly support concurrency without having to use API. [17] The concurrent activities are called threads in Java [17][18][21] and C# [23] while called tasks in Ada. Different thread and tasks can be executed in parallel on different cores of a multi-core processor.

3. GPU-equipped Personal Computers

Compute Unified Device Architecture (CUDA) is an architecture developed by NVIDIA for its GPUs. [27][28][29] CUDA adopts many-core approach which has numerous much smaller cores than the cores of a multi-core CPU. The massive GPU cores are optimized for floating-point calculations while the CPU cores are optimized for sequential code execution. [28] Engineering applications rely heavily on floating-point calculations. CUDA can place much more cores in a chip than multi-core CPUs. That makes it more powerful than multi-core CPUs in floating-point calculations as much more cores are dedicated to the purpose. In 2009, the ratio for peak floating-point calculation throughput between many-core GPU and multi-core CPU is about 10 to 1. [28] The Chinese Tienhe-1A supercomputer took the title of fastest supercomputer in the world from American Jaguar supercomputer in November 2011 with 37794 less CPU cores by adopting GPUs. GPUs' design philosophy also makes them cheaper than multi-core CPUs. Table. 1 lists the numbers of GPU cores and costs of four different GPU cards. The numbers of CPU cores and costs of three different CPUs are shown in Table 2. Those prices as of October 2011were found on www.pricewatch.com. Apparently, the cost/performance ratio of GPUs is much better than that of multi-core CPUs. The programming language is CUDA C/C++ which is based on C/C++. Lots of computational scientists and researchers are adopting GPU programming because it is very cost-effective. Therefore, GPU computing is very promising and should be covered in engineering education.

Model	Number of GPU Cores	Price
GeForce 210	16	$35
GeForce GT 430	96	$60
GeForce GTX 460	336	$149
GeForce GTX 580	512	$460

Table. 1 Features and Costs of GPU

Model	Number of CPU Cores	Price
Intel Core i7 3.06GHz	4	$297
AMD Phenom II X6 3GHz	6	$169
AMD Phenom II X4 3GHz	4	$120

Table. 2 Features and Costs of CPU

With the low-costs of GPUs, PCs equipped with GPU cards are affordable for any institution. The CUDA library and compiler is provided by NVIDIA for free for Linux, Windows, and MacOS. All software required for GPU programming is free if Linux platform is adopted.

3.1 Hardware Requirement

Any graphic card equipped with NVIDIA GPU can be utilized. GPU Device drivers have to be installed and can be downloaded from www.nvidia.com. [27]

3.2 Software Requirement

The software required for GPU programming is C compiler for GPU and C compiler for CPU. The GPU compiler is contained in CUDA Development Toolkit which is freely downloadable at NVIDIA's web site. [27] The GNU C compiler coming with Linux can be used as the compiler for CPU without charge.

4. Commodity Clusters

C/C++ and FORTRAN has dominated the numerical methods field, a key part of computational science, for decades and numerous of programs were coded in both languages. It is beneficial to stick with both languages and reuse existing codes and libraries. Message Passing Interface (MPI) is a standard developed for parallel libraries supporting C/C++ and FORTRAN.[1][2][3][10] [25][26] Several MPI implementation has been developed for different platforms. MPI programs are portable among those platforms. Message-passing clusters are the most popular high performance computers. Students learning MPI programming from small-scale commodity clusters can easily adapt to large-scale high performance clusters. Thanks to the contribution of open-source software developers, MPI libraries have been successfully ported to inexpensive PC platform. Along with other free open-source operating systems and applications for PC, they can make PCs networked by low-cost switches a commodity cluster, an affordable platform for high performance computing education. With clusters built with the retired PCs and free software, any institution can own its platforms with minimal cost and start high performance computing education. Although clusters built with retired PCs do not have sufficient computing power to execute large-scale parallel applications, they do exhibit all characteristic of parallel processing and can execute qualitative experiments. If an institution owns sufficient funding, it can acquire high-end PCs and construct a cluster which has decent computing power to execute serious parallel programs for research. While an affordable cluster can be built with the free open-source software and retired PCs, it cannot work practically without some vital configurations. Several books have been written about building a cluster with Linux. [4][5][6][7][8] However, all of them fail to point out the vital configurations required to make the cluster work correctly. The practical issues in building an inexpensive cluster are addressed in the following subsections respectively.

4.1 Hardware Requirements

Any PC with 128B RAM or more and an Ethernet network interface card (NIC) can work as a node. PCs with Pentium III 500MHz CPUs work smoothly at author's institution. All nodes have to be connected with a network switch. Generic Ethernet NICs and switches can be acquired with very little cost. The logical layout is shown in Figure 1 and a 10-node commodity cluster is displayed in Figure 2.

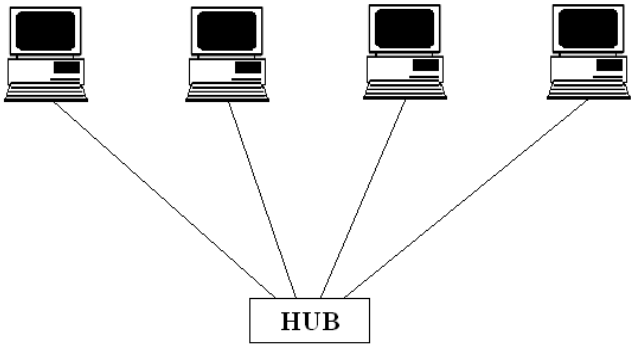

Fig. 1 Logical layout of a commodity cluster

Fig. 2 A 10-node commodity cluster built of retired PCs

4.2 Operating System and Software Packages

Among all operating systems, only Linux can be acquired for free. Also, Linux operating systems come with plenty of hardware drivers which cover almost all legacy and new hardware, that further makes it the ideal OS for commodity clusters. Although several implementations of open source Linux operating systems are available, not all of them work well with the MPI library. After extensive experiments, the author chose and installed CentOS Linux which is a clone of commercial Red Hat Enterprise Linux [11] and downloadable at www.centos.org. The following actions should be done for all nodes in the cluster. Firewall and SELinux should be turned off during installation as they cause difficulty for communications among nodes which are required for executing MPI programs. Fortunately, security is not a concern as long as the cluster is not connected with other networks. The editor, GNU C++ and FORTRAN compilers under "Development Tools" have to be installed to as they are required for MPI programming process. One node should be selected as the server node and the following software should be installed on the node during installation: Network Information Service (NIS) server under "Network Server" and Network File System (NFS) server configuration tool under "Server Configuration Tool." After Linux is installed, networking should be correctly configured so that all nodes of the cluster can communicate among one another. In general, one user account with the same name needs to be created on each node because a parallel program is dispatched to each node under the same user account. NIS which is addressed in later subsections can take care of this and other issues.

4.3 MPI library

The next major step is to install the MPI library. Again, there are several free implementations of MPI, such as MPICH, LAM/MPI, Open MPI, etc. Nevertheless, only Open MPI is still under active development and growing more powerful. [9] Therefore, Open MPI is the best choice. The steps to install Open MPI are quite straightforward. They are described as follows. First, download the latest version of the library from Open MPI's website, www.openmpi.org. Log into the root account to install it. Copy the compressed file to the /tmp directory. Uncompress the file by double clicking the file icon to uncompress the library. Then change to the directory openmpi-1.4.3. Configure and make Open MPI with the commands below. [9] Replace the directory after prefix option if you want to install into another directory. The make process may take up to one hour on an old PC.

```
shell$ ./configure - -prefix=/usr/local
shell$ make all install
```

4.4 Need for NFS and NIS

Before we run a parallel program on our cluster, we need to dispatch a copy of the program's executable file onto every node under the same account. Manually copying the executable to all nodes is impractical. Network File System (NFS) is the solution to this requirement. With NFS, the program's executables only need to be saved into the shared directory of the NFS and a copy of the program will be automatically copied to all other nodes. It is also impractical to create accounts for all users on all nodes. To remedy this problem, Network Information System (NIS) is used to create accounts on the server node. After all nodes are configured as NIS clients, all users can login from any node and run their own parallel programs. You have to login as root to perform all following setups and reboot all nodes to take effect. Do not reboot any client node until the server node completes its boot-up process, otherwise client nodes cannot read the correct configuration information from the server and perform normally.

4.5 Set Up the NFS Server [11]

1) From the *NFS Server Configuration* window, click *File → Add Share*. The *Add NFS Share* window appears. In the *Add NFS Share* window Basic tab, type the following information:
    - Directory – Type the name of the directory you want to share. Type "/home" which is the parent directory to all user directories.
    - Host(s) – Enter one or more host names to indicate which hosts can access the shared directory. Type "*" to let all nodes access NFS server.
    - Basic permissions – Click *Read/Write* to let remote computers mount the shared directory with read/write access.

2) To permanently turn on the NFS service, type:

```
shell$ chkconfig nfs on
shell$ chkconfig nfslock on
```

4.6 Set Up the NFS Client [11]

To set up an NFS file system to mount automatically each time you start your Linux system, you need to add an entry for that NFS file system to the /etc/fstab file. The /etc/fstab file contains information about all different kinds of mounted file systems for your Linux system. The format for adding an NFS file system to your local system is the following:

host:directory mountpoint options 0 0
The first item identifies the NFS server computer and shared directory. Mountpoint is the local mount point on which the NFS directory is mounted, followed by the file system type nfs. Any options related to the mount appear next in a comma separated list. For our system, we add the following NFS entries to /etc/fstab:
kingtiger1:/home /home nfs rsize=8192,wsize=8192 0

4.7 Set Up the NIS Client [11][12]

All nodes have to be configured as NIS clients by the following steps. Even the NIS server has to be set up as an NIS client first.

1) Defining an NIS domain name:
   To make the NIS domain name permanently, you need to have the domainname command run automatically each time your system boots. It can be done by adding the command line to a run-level script that runs before the ypbind daemon is started. The following line should be added just after the first set of comment lines in the /etc/init.d/network file.
   domainname kingtiger
2) Setting up the /etc/yp.conf file:
   We have an NIS domain called kingtiger and a server named kingtiger1, the following entries are added in the /etc/yp.conf file:
   domain kingtiger server kingtiger1
   domain kingtiger broadcast
   ypserver kingtiger1
3) Configuring NIS client daemons:
   We need set up an existing run-level script called ypbind to start automatically at boot time. To do this, run the following command:
   shell$ chkconfig ypbind on
4) Using NIS maps:
   For the information being distributed by the NIS server to be used by the NIS clients, you must configure the /etc/nsswitch.conf file to include nis in the search path for each file you want to use. In most cases, the local files (files) are checked first, followed by nis. The following are examples of how some entries should be changed:
   passwd: files nis
   shadow: files nis
   group: files nis
   hosts: files nis dns

4.8 Set Up the NIS Server [11][12]

1) To configure your Linux system as an NIS server, you should first configure it as an NIS client and reboot the system.
2) Creating NIS maps:
   To create NIS maps so that your Linux system can be an NIS server, start from the /var/yp directory from a Terminal window as root user. In that directory, a Makefile enables you to configure which files are being shared with NIS. All default configurations in Makefile are ok for our purposes, so we don't need change them.
3) Configuring access to maps:

In the /etc/ypserv.conf file, you can define rules regarding which client host computers have access to which maps. For our purposes we just need add the following line into /etc/ypserv.conf to allow all hosts access to all maps:

```
*  :  *  :  *  :  none
```

4) Configuring NIS server daemons:
   We can use the following chkconfig command to set ypserv and yppasswdd scripts to start automatically at boot time.
   shell$ chkconfig ypserv on
   shell$ chkconfig yppasswdd on
5) Updating the NIS maps:
   If you modify the sources for NIS maps (for example if you create a new user by adding the account to the passwd file), you need to regenerate the NIS maps. This is done by a simple
   make –C /var/yp
   This command will check which sources have changed, creates the maps new and tell ypserv that the maps have changed.

4.9 Disabling Password Authentication

As Open MPI is configured by default to use ssh (secured shell) to dispatch parallel tasks, it is ssh that asks for password to authenticate the connection. Extra steps below will prevent ssh from requesting passwords [9]. Because the measure should only work for the user account which intends to run MPI applications, log into the specific user account instead of root account to configure. First, generate the private and public key for the user account by executing:
shell$ ssh-keygen -t dsa
That will generate the hidden .ssh directory with the necessary attribute. Change into the .ssh directory and do the following. [9]
shell$ cp id_dsa.pub authorized_keys
With these procedures done, the public keys are duplicated as the authorized keys. They will be used for authentication for all future connections without passwords being requested from other nodes. Now the MPI applications can be executed on multiple nodes of this cluster without being asked for passwords.

5. Conclusion

With the maturity and availability of high performance computers, numerical simulation has surpassed experimentation and become the most important tools for research and design as it is very cost effective. Most engineering applications utilize numerical simulation to resolve problems. In turn, the efficiency of numerical simulation depends on high performance computing. That fact necessitates engineering education to cover high performance computing. As the development of processors has shifted to multi-core approach, parallel programming becomes imperative to take advantage of the extra cores of modern CPUs. Likewise, that demands the inclusion of parallel programming in engineering education. Nonetheless, the challenge for most institutions to offer both topics in engineering education is the high cost of high performance computers. The author explored and managed to discover three approaches to build affordable high performance computing platforms for high performance computing education. The first approach is to adopt PCs equipped with multi-core processors and install multithreading libraries. Multithreaded programs which exhibit parallelism can be executed on all cores of the multi-core processor. Students can learn parallel programming through writing the multithreading programs. This approach does not require any extra hardware or configuration to build the high performance computing platform. The second approach is to adopt the emerging GPU programming by acquiring PCs equipped with inexpensive GPU cards. This approach requiring only insertion of GPU cards into the PCI

Express slots in the PCs and installation of software drivers. The third approach is to build commodity cluster with PCs and network switches. This approach needs some system configurations which are all described in this paper. As retired PCs can be used and all required software can be downloaded for free, this approach is the cheapest way to build high performance computing platforms. While the three approaches may not have the computing power to carry out large scale parallel applications, all of them exhibit full features of parallel systems and applications. Hence, they are ideal for teaching high performance computing in engineering education.

References

[1] Quinn, Michael J., Parallel **Programming in C with MPI and OpenMP**, McGraw Hill, 2004
[2] Karniadakis, George and Kirby, Robert, **Parallel Scientific Computing in C++ and MPI**, Cambridge University Press, 2003
[3] Grama, Ananth et al., **Introduction to Parallel Computing**, Addison-Wesley, 2003
[4] Sloan, Joseph D., **High Performance Linux Clusters**, O'Reilly, 2005
[5] Bookman, Charles, **Linux Clustering**, New Riders, 2003
[6] Vrenios, Alex, **Linux Cluster Architecture**, Sams, 2002
[7] Lucke, Robert, **Building Clustered Linux Systems**, 2005
[8] Gropp, William et al., **Beowulf Cluster Computing with Linux**, 2nd Ed., MIT Press, 2003
[9] www.openmpi.org FAQ
[10] Pacheco, Peter S., **Parallel Programming with MPI**, Morgan Kaufmann Publishers, Inc. 1997
[11] Sobell, Mark, **A Practical Guide to Red Hat Linux**, Third Edition, Prentice-Hall, 2006
[12] http://www.linux-nis.org/nis-howto/
[13] Hughes, C. and Hughes, T., **Professional Multicore Programming**, Wiley, 2008
[14] Hughes, C. and Hughes, T., **Parallel and Distributed Programming Using C++**, Addison-Wesley, 2004
[15] Reinders, James, **Intel Threading Building Blocks**, O'Reilly, 2007
[16] Nichols, B. et al., **Pthreads Programming**, O'Reilly, 1996
[17] Welling, A., **Concurrent and Real-Time Programming in Java**, Wiley, 2004
[18] Breshears, C., **The Art of Concurrency**, O'Reilly, 2009
[19] Parhami, B., **Introduction to Parallel Processing**, Plenum, 1999
[20] Buyya, R., **High Performance Cluster Computing**, Vol. 2, Prentice-Hall, 1999
[21] Magee, J. and Kramer, J., **Concurrency**, Wiley, 2006
[22] Yang, L. and Guo, M., **High-Performance Computing**, Wiley, 2006
[23] Arora, G. et al., **Microsoft C# Professional Projects**, Premier, 2002
[24] http://software.intel.com/en-us/blogs/2006/10/19/why-windows-threads-are-better-than-posix-threads/
[25] Gropp, W. et al., **Using MPI**, MIT Press, 1999
[26] Gropp, W. et al., **Using MPI-2**, MIT Press, 1999
[27] Sanders, J. and Kandrot, E., **CUDA by Example**, Addison-Wesley, 2011,
[28] Kirk, D. and Hwu, W., **Programming Massively Parallel Processors**, Morgan Kaufmann, 2010
[29] Hwu, W., **GPU Computing Gems Emerald Edition**, Morgan Kaufmann, 2011
[30] www.top500.org