High-performance Computing and Virginia Tech "System X"

David Benham and Yu-Chung Chen

Department of Computer Science

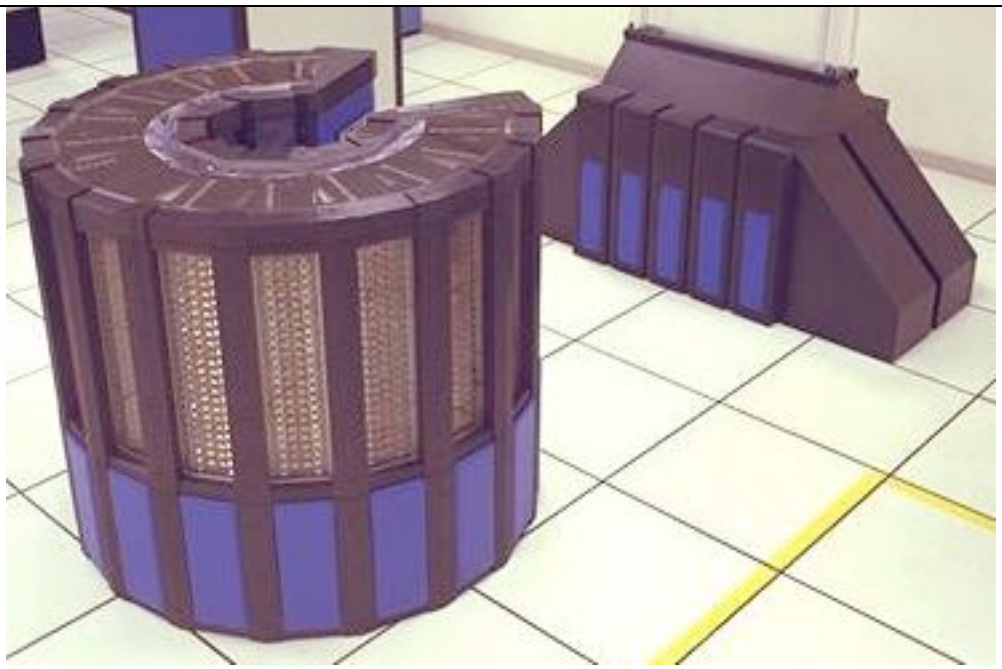
28 November 2005

Abstract

This project is the research of Virginia Tech's Terascale facility, also known as "System X". This report is for CS 466 Advanced Computer Architecture class taught in Fall 2005. In this report we present our research on the ideas of high-performance computing and the system aspect of the G5-based "System X". Section 2, 3, 4, 6 are from David Benham and Section 1 and 5 are from Yu-Chung Chen.

1. High-performance computing

The Top500 list [1] is compiled twice a year by Jack Dongarra of the University of Tennessee, Knoxville; Erich Stohmaier and Horst Simon of NERSC/Lawrence Berkeley National Laboratory; Hans Meuer of the University of Mannheim, Germany [2]. The latest list is released in Nov. 2005 Supercomputing conference, an annual conference held specifically for areas of high-performance and supercomputing. Unlike consumer or business computing needs, high-performance computing (HPC) demands on performance so much that system architects have to squeeze out the best performance. In the old days, supercomputers are almost all specially made machines. Unlike commodity off-shelf computers, some of them are designed to work on complicated mathematics and scientific problems in an efficient way, for example vector-based supercomputer Cray series [3]. And most of them will cost up to several million dollars.



Cray-2 supercomputer http://en.wikipedia.org/wiki/Image:Cray.jpeg [4]

But in the past decade, commodity computer components or personal computers are taking off like never before. More and more systems appearing on the top500 list are made with a large number of personal computers connected together as a 'PC cluster' or 'Beowulf PC cluster'. In today's latest top500 list, there are 360 systems being defined as clusters, making it the most common architecture [2].

1.1 Hardware and software stacks

To build a cluster of computers, making it work as one system, the system architect must consider hardware and software components required. Here we will take a look at some basic required components of a working cluster system.

In hardware aspect, a computational cluster will typically have huge number of CPUs partitioned in computation 'nodes'. Take modern personal computer for example: one node might contain 1 to 4 CPUs. Each node has their local memory and local scratch disk shared by multiple CPUs (if there's more than 1). Each node is connected together

with high-speed networking. Typically we will call this kind of setup is a “distributed memory” [6]. There are also systems using “distributed shared memory” [7].

In the software side, since most of the target user of such high computation power systems are application scientists and researchers, the operation systems used are more U*IX based than commodity operation systems like Microsoft Windows. Currently the most popular ones are Linux [8] or BSDs [9].

Aside from the basic operation system, to meet the requirements of running computation codes across multiple computer systems (nodes), management and programming middleware are also developed. In the management software, due to cluster systems are to be fully utilized, scheduling and queuing software systems are developed for the system administrators. Users must submit their computation ‘jobs’ through queuing system along with the job description information. Then the scheduling middleware will look at the job’s specification and then try to allocate required resources for the job. When the resources required are available, then the job will be put into execution. Currently most widely used and freely available software are Torque resource manager [10] (former Portable Batch System [11]) and Maui scheduler [12].

For parallel computing middleware, there are several programming paradigms are proposed along the development of supercomputering history. Also because most of the users of computational cluster system might not be experts of computer for information science background. Most of them are researchers and scientists in different application domains like civil engineering, computational fluid dynamics or micro-electro-mechanical systems. Providing them a higher level of programming paradigm will model the whole system in a more abstract level and ease the learning curve of inside- outs knowledge of the whole system dramatically. OpenMP [13] is an application program interface (API) for distributed shared memory. Due to some old supercomputer are distributed shared memory based, like Silicon Graphics, Inc. Origin 3000 [14], OpenMP is quite common for scientists who used to work on large shared memory machines. Parallel Virtual Machine (PVM) is developed by University of Tennessee, the Oak Ridge

National Laboratory and Emory University. PVM provides the programmer an abstract view over the whole cluster as one massive parallel computer, which ease the tedious details of computer architecture and system setups. The last one and also the most important one middleware nowadays is "Message-passing Interface (MPI)" [16]. MPI is first developed in Mathematics and Computer Science Division, Argonne National Laboratory [17]. Later it is "proposed and standardized by a broadly based committee of vendors, implementers, and users" [16].

1.2 Applications

With so many computers and so much computing power, what can a computation cluster or supercomputer be used for? We will first look at the system level and then the application level. Furthermore, in application level, we will look at different application domains ranging from tightly coupled applications like computational fluid dynamics to loosely coupled applications like SETI@Home.

In the system level, a large number of computers can served as a server farm to providing enterprise infrastructure needs. The need of enterprise for server farms can be categorized into two main functionalities: "high availability (HA)" and "high throughput". "High availability" means the time of one service available to the user should be high and the time the service is down should be low. With proper software and hardware technology, if one service provided by a node is down, the system should be able to allocate another node and switch the user's service to another node and the end user is not aware. "High throughput" means the number of incoming user requests one system can serve at a period of time. If we assume that the incoming services requests are independent with each other, with increasing number of nodes in a cluster computing system, the throughput number of the system can serve will be nearly linear proportional to the number of nodes available in the system. Almost most of modern enterprise system will employ such system clustering to provide better customer services. One outstanding example is Google complex service farms [18]. According to [18], Google has several clusters geographically distributed worldwide. Each cluster contains a few thousand mid-range desktop x86-based machines. All hooked up with reliability-focused software

system provide the huge amount of requests and computing needs of the whole Google services.

In the computational-oriented level, the applications can be categorized into two kinds, tightly coupled and loosely coupled applications. In tightly coupled applications, the parallel computing is distributed to all computing nodes and the computations happened in a 'near real time' fashion. When a computing job submitted to a system, the 'job master' would try to spawn multiple 'job slaves' in nodes (resource) allocated by the resource manager. Depends on the computation needs, the slave jobs in different nodes might communicate and exchange data with each other (including master). Take CFD application for example [19], if the shape of a louvered fin cannot fit into a single computation node, it can be partitioned into multiple grids and distributed across multiple computation nodes. Each grid components in single computation node is actually related to each other, so the "boundary condition" must be considered. In some areas, the calculation results must be passed to the next block; also the calculation in the next might not be able to start until other nodes' results arrive. This kind of parallel programming paradigm is suitable for application like computational fluid dynamics (CFD) and micro-electro-mechanical systems (MEMS). Those problems tend to be too huge for a single machine and the communication between cluster nodes is frequent. In the loosely coupled applications, it is also called "embarrassingly parallel" [20]. In this kind of parallelism, the computation codes in different nodes will run independently from each other. This also means the communications between each slave to the master or between slaves are minimal. The best application suited for this kind of parallelism is "single program multiple data" program. One of the most the most famous examples is SETI@Home [21].

2. Overview of System X

The Virginia Tech System X supercomputer was once held the distinction of being the 7th fastest computer in the world. But that was last year. As of the date of the writing of this paper, Virginia Tech's System X is now ranked 20th in the world [4]. The

rank is a noteworthy distinction, regardless of which figure is used. The ever changing list of the most powerful computers on the planet demonstrates just how quickly advances in hardware and software allow high performance computing to make continuous and significant advances in performance.

Illustration 1: System X [1]

The first version of System X was conceived in January of 2003. At a cost of 5.5 million dollars, the initial System X computer was composed of 1100 Power Mac G5 desktop computers [2]. However, the memory used in the G5 computer contained no logic for correcting errors, so the ability to use System X for scientific computations was limited [2].

To correct this deficiency, System X underwent a significant upgrade in 2004. Over the course six months, the Power Mac G5 machines were all replaced with custom Xserve G5 servers for a total cost of $600,000. The Xserve servers had a 15 percent performance increase over the G5 desktop machines. The Xserve also brought error correcting code (ECC) memory to System X and significantly reduced the physical amount of space required to house the hardware [3]. The upgrade, along with some software optimizations, boosted the performance of System X by 20 percent.

Current System X system specifications obtained from [1]:

- **Benchmark performance** - Linpack Rpeak = 20.24 Teraflops, Rmax = 12.25 Teraflops
- **Nodes**
  - **Compute nodes -** 1100 Apple XServe servers. Dual 2.3 GHz PowerPC 970FX processors, 4 GB ECC DDR400 (PC3200) RAM, 80 GB S-ATA hard disk drive, Mellanox Cougar InfiniBand 4x HCA
  - **Compile nodes -** 3 Apple XServe servers. Dual 2.3 GHz PowerPC 970FX processors, 4 GB ECC DDR400 (PC3200) RAM, 3x250 GB S-ATA hard disk drive
- **Network**
  - **Primary -** Infiniband
    - 4 SilverStream Technologies 9120 Infiniband core switches. 4X Infiniband, 10 Gbps bidirectional port speed, 132 ports per switch
    - 64 SilverStream Technologies Infiniband leaf switches. 4X Infiniband, 10 Gbps bidirectional port speed, 24 ports per switch
  - **Secondary -** Gigabit Ethernet. Provided by 6 Cisco 4506 Gigabit Ethernet switches.
- **Storage -** Apple XServe RAID. Total storage: 2.7TB
- **Software**
  - Mac OS X 10.3.9
  - MVAPICH for message passing
  - IBM XL Fortran, IBM XLC, and gcc
  - Torque (Queue) and Moab(Scheduler)

3. Dollars and Cents

System X, aside from being a pioneer in terms of raw computing power, is possibly even more impressive when analyzed from a cost standpoint. At the time of construction, the total cost for the system was 5.7 million dollars, which was by far the lowest cost of any comparable machine. In fact, when System X was #4 in the top 500 list in 2004, it was constructed for only 20 percent of the cost of the next least expensive machine in the top 10 [1].

In addition to hardware costs, operating costs of System X played a critical factor during its construction. When operating thousands of nodes, every watt matters. The PowerPC 970FX consumes significantly less power (55 watts) when compared to the AMD Opteron (89 watts) or an Intel Xeon (110W) [2]. As a whole, System X consumes 310 kilowatts of power or about 250 MW/month.

4. System Cooling

Transferring the heat from System X to the outdoor water chillers begins inside the individual nodes themselves. Apple Xserve servers are air-cooled and contain two air ducts on the front of the machine and a rear mounted fan to obtain a front to rear airflow [1]. Each rack contains air to liquid heat exchangers that take heated air and transfer the heat to a liquid refrigerant. Each rack is connected to a Liebert Extreme Density XDP unit, which transfers heat from the refrigerant to chilled water loops [2]. This setup helps isolate chilled water loops and greatly decreases risk associated with leakage in piping and reduces risk of damage to equipment [2]. Essentially, the only water that exists in the entire system is contained in these units. Once the heat is removed from the refrigerant, the water is pumped to outdoor water chillers.

Illustration 2: Liebert Extreme Density XDP unit [2]

At the end (or beginning, depending on your perspective) of the cooling system for System X are two 125 ton Carrier water chillers. Together these two units can provide

3 million BTUs of cooling capacity. System X utilizes just under half of its available cooling capacity.

Illustration 3: Outdoor Carrier water chillers [2]

5. Interconnection

In the networking part, System X uses InfiniBand as the major internal message passing fabric. Each computation node is equipped with an InfiniBand host channel adaptor (HCA). Nodes are first connected to 64 InfiniBand switches and the uplink to 4 core switches backbone [22], to construct a "fat-tree" [27-29] overall architecture. Also the built-in gigabit Ethernet ports are used for typical administration and job submission purpose. The following table is downloaded from [22], lists System X's major network components.

Primary	4 SilverStorm Technology 9120 InfiniBand core switches	- 4X InfiniBand, 10 Gbps bidirectional port speed. - Each Switch populated with 11 leaf modules and 3 spine modules - Total 132 InfiniBand ports per core switch
	64 SilverStorm Technologies 9024 InfiniBand leaf switches	- 4X InfiniBand, 10 Gbps bidirectional port speed - Total 24 InfiniBand ports per leaf switch
	InfiniBand fabric management by SilverStorm Technologies	
Secondary	6 Cisco Systems 240-port 4506 Gigabit Ethernet switches	
Table data courtesy of Virginia Tech Terascale Computing Facility [22]		

5.1 About InfiniBand architecture

With the computing power advances, the CPUs are gaining more and more power over time following the Moore Law [30]. For example, one first generation G5 CPU has two double-precision floating-point units, meaning 2 floating-point operations per cycle. In a 2GHz CPU, it means 8 Gflops (Giga-FLoating-points Operations Per second). With dual G5 CPUs, one node can achieve 16 Gflops peak performance! But on the other end, the input/output system is always much slower than the computation core. Even worse, with the advances in CPU, the gap between could get even bigger over time.

"InfiniBand is a high performance, switched fabric interconnect standard for servers" [31]. InfiniBand was first designed for storage area network. But due to its high bandwidth, low latency characteristics, more and more cluster architects are using InfiniBand as the interconnect when building clusters and supercomputers. Merged from two competing infinitives designing a better I/O system for the computer system in the future [32], InfiniBand absorbs merits from both camps. To become the I/O standard of the future, performance, flexibility, scalability, and reliability are all being taken into account in the specification design phase.

A basic single IB channel connection can have 5 Gbps bidirectional bandwidth. IB also supports double and quad data rates for 10 Gbps and 20 Gbps respectively. With 8B/10B encoding [32], about 4/5 of raw data stream are actually user data. Theoretically multiple IB links can be aggregated together like Ethernet channel bonding does to achieve high bandwidths.

Unit: Gbps	SDR	DDR	QDR
1X	2.5	5	10
4X	10	20	40
12X	30	60	120
Table information downloaded from [32] (Uni-direction bandwidth in listed, and this is 'raw' data size, not user data size.)			

Just like other new interconnect network technologies like Myrinet [35] and Quadrics [36], all of them do not use TCP/IP protocol natively. One reason is that the TCP/IP stack would pose too much latency. Data stream need to be transferred from one end going all the way down to the bottom of the stack and then sent through the network fabric and unpack all the way up at the other end of TCP/IP-based network. Not using TCP/IP protocol implies that some of the communication middleware might need to be rewritten, like widely used message passing interface, MPI. Currently MVAPICH – (MPI for InfiniBand on VAPI) [37] is publicly available for InfiniBand-based cluster system to gain the best performance. And according to [37], it powers up to 4000-node (8000-processors) InfiniBand cluster at Sandia National Laboratory [2]. As we can see since from 2003 when System X is built with first system using InfiniBand in top500 list, InfiniBand network architecture is getting more and more popular in HPC to obtain better performance.

5.2 Infiniband applications

Except high-performance supercomputers, the high bandwidth and low latency characteristics of InfiniBand also fit to the requirement of parallel visualization system. In the recent talk in OpenIB alliance meeting [38], Stanford University uses cheap PC clusters to outperform more traditional graphics supercomputers [39]. Aside from all the advantages PC cluster can provide, the interconnection using InfiniBand also plays an important role, especially in the interaction required applications. For example in volume rendering, the user would expect to manipulate a volume data of size 1K * 1K * 1K with reasonable frame rate. With the results from Stanford SPIRE graphics cluster, they can achieve performance of 8.2 GVoxel/s (1024^3 at 8 frames per second) at 790 MB/s node to node and 10.2 GB/s cross sectional bandwidth [38, A8 – Visualization Experience and Requirements]. In the future, IB could skip computer system's internal bus (PCI-X or PCI-Express), hooking up "HyperTransport" technology used in AMD etc [32, 34] through "HyperTunnel", to build a massive computing system with high speed CPUs and high bandwidth, low latency I/O system. And it is also scalable, flexible and reliable.

6. Message Passing

Communication between processes on System X is accomplished using MVAPICH. MVAPICH is an implementation of the Message Passing Interface (MPI) and MVAPICH is essentially an "offshoot" of the popular MPICH, an implementation of MPI maintained by Argonne National Laboratory [2].

A number of implementations of MPI exist, often the choice of what implementation to use is influenced by interconnect technology and platform.

For Mac OS X, a number of MPI implementations options exist [2].

- **MPICH** - One of the oldest MPI implementations available.
    - **MVAPICH -** Implementation that utilizes the Infiniband interconnect technologies.
- **LAM/MPI** - Maintained by Pervasive Technology Labs at Indiana University. Offers a native installer in lieu of a UNIX tar package.
- **MacMPI** - Fist MPI implementation for the Macintosh platform. Presents a graphical representation of node status.

In MPI, each process is assigned a positive number starting from zero. So x processes will be numbered from 0 to x-1. Perhaps the quickest way to introduce the basics of MPI is to start with the classic "Hello World!" [3] . The following code is from [1] and is modified slightly for clarity:

```
1       #include <stdio.h>
2       #include "mpi.h"
3
4       main(int argc, char** argv) {
5               int my_rank;            /* Rank of process */
6               int p;                  /* Number of processes */
7               int source;             /* Rank of sender */
8               int dest;               /* Rank of receiver */
9               int tag = 50;           /* Tag for messages */
10              char message[100];      /* Storage for the message */
11              MPI_Status status;      /*Return status for receive */
12
13              MPI_Init(&argc, &argv);
14              MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
15              MPI_Comm_size(MPI_COMM_WORLD, &p);
16
```

```
17              if (my_rank != 0) {
18                      sprintf(message, "Hello from proc %d!", my_rank);
19                      dest = 0;
20
21                      /* Use strlen(message)+1 to include '\0' */
22                      MPI_Send(message,strlen(message)+1,
23                              MPI_CHAR, dest, tag, MPI_COMM_WORLD);
24              }
25              else { /* my_rank == 0 */
26                      for (source = 1; source < p; source++) {
27                              MPI_Recv(message, 100, MPI_CHAR, source, tag,
28                                      MPI_COMM_WORLD, &status);
29                              printf("%s\n", message);
30                      }
31              }
32              MPI_Finalize();
33      }
```

In this program, the process with ID 0 receives a simple “hello” message from all the other processes running this program. Each processor in a multiprocessor system will receive a copy of this program and execute it. Depending on the identity of the process, a different section of code will be executed. This is a *Single Program Multiple Data or SPMD model*. In this model, a different portion of the same program is executed depending on the data local to an individual process. Our simple example has a single branch and each process will take either one branch or the other. Each process executes it's own stream of instructions based on conditional branches in a single program. This approach is often used when writing *Multiple Instruction Multiple Data or MIMD* programs using MPI [1].

In our example, we called a handful of MPI functions:

- **MPI_Init** - Executes any code the system needs to begin executing MPI function calls. This function must be called before any other MPI function.
- **MPI_Comm_rank** - Determines the rank of a given process.
- **MPI_Comm_siz**e- Returns the total number of processes executing a program.
- **MPI_Send** - Send a message to a specified process with a known rank. Function will block, program execution will not continue until the message is sent.
- **MPI_Recv** - Receive a message from a specified process with a known rank. Function will block, program execution will not continue until a message is

received.

- **MPI_Finalize** - Execute code to clean up.

Our “Hello World!” program, while it might help illustrate simple MPI concepts, is actually a fairly bad example of parallel programing for several reasons. First, the work is not evenly distributed. Every process reports to process 0, so process 0 is doing a disproportionate amount of work in the program. Secondly, by using blocking send and receive calls, programs spend a great deal of time doing nothing waiting for other processes to respond or send information. There are ways to address both of these issues using MPI.

Ideally, non blocking communication functions should be used in an MPI program. MPI provides non blocking communication functions:

- **MPI_ISend()** - non blocking send operation, allows program execution to continue
- **MPI_IRecv()** - non blocking receive operation, allows program execution to continue
- **MPI_Wait()** - completes the communication transaction

In addition to non blocking communication functions, the use of broadcast communication can greatly simplify communications in an MPI program. Imagine you need to pass runtime information to a group of processes. Unless you have prior knowledge of the number of processes running in the system, you will have to loop through and pass the information to each process running the program. Instead, MPI gives you the function **MPI_Bcast**, which in a single call, will broadcast messages to groups of processes.

6. 1 Can't we all work together?

MPI also provides communication functionality designed aiding the programmer accomplish collaborative tasks. Many tasks which are broken down must often combine or compare their results to derive a final solution to a problem.

For example, if you were computing of a integral over an interval [a ,b], one approach could assign each process in a multiprocessor system the task of solving a subinterval of the integral. Once each process completed, coming up for a value for the original integral would simply require adding all the calculations from each process that

was assigned a portion of the integral. MPI provides a single function to aid in calculations of this type, **MPI Reduce** [1]. Every process in the system would call this function, and processes responsible for collecting the results would neatly assemble their figures. The **MPI_Reduce** function provides the capability for the following collaborative calculations [1]:

Operation	***Meaning***
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical And
MPI_BAND	Bitwise And
MPI_LOR	Logical Or
MPI_BOR	Bitwise Or
MPI_LXOR	Logical Exclusive Or
MPI_BXOR	Bitwise Exclusive Or
MPI_MAXLOC	Maximum and Location of Maximum
MPI_MINLOC	Minimum and Location of Minimum

Other collaborative MPI functions include [1]:

- MPI_Barrier - allows multiple processes to synchronize at a given point in a program
- MPI_Gather - Collect data from multiple processes reporting to a given root

- MPI_Scatter - A root process can distribute a buffer of information to other processes.
- MPI_Allgather - Similar to MPI_Gather, but when operating on a group of processes, gives each process data collected from all the other processes in a group
- MPI_Allreduce - Similar to MPI_Scatter, but when operating on a group of processes, sends each process in the group information from every other process

Bibliographies

Sec. 1 – High-performance computing

Sec. 5 – Interconnection

1. Top 500 supercomputer sites: http://www.top500.org/
2. Top 500 list for November 2005: http://top500.org/lists/2005/11/
3. Cray Inc.: http://www.cray.com/
4. Wikipedia – Cray: http://en.wikipedia.org/wiki/Cray
5. Wikipedia – Distributed memory: http://en.wikipedia.org/wiki/Distributed_memory
6. Wikipedia – Distributed shared memory: http://en.wikipedia.org/wiki/Distributed_shared_memory
7. Wikipedia – Beowulf: http://en.wikipedia.org/wiki/Beowulf_%28computing%29
8. Linux: http://www.linux.org/
9. Wikipedia – Berkeley Software Distribution: http://en.wikipedia.org/wiki/BSD
10. Torque resource manager: http://www.clusterresources.com/pages/products/torque-resource-manager.php
11. Portable Batch System: http://www.openpbs.org/
12. Maui scheduler: http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php
13. OpenMP: http://www.openmp.org/
14. Silicon Graphics Origin 3000: http://www.sgi.com/products/servers/origin/3000/
15. Parallel Virtual Machine: http://www.csm.ornl.gov/pvm/
16. Message Passing Interface: http://www-unix.mcs.anl.gov/mpi/
17. Mathematics and Computer Science Division, Argonne National Laboratory: http://www-unix.mcs.anl.gov/
18. “Web Search for a Planet: the Google Cluster Architecture”: http://labs.google.com/papers/googlecluster-ieee.pdf
19. NCSA CFD story: http://www.ncsa.uiuc.edu/News/Access/Stories/CoolTurbines/
20. Wikipedia – Embarrassingly parallel: http://en.wikipedia.org/wiki/Embarrassingly_parallel
21. SETI at home: http://setiathome.ssl.berkeley.edu/

22. Virginia Tech Terascale Computing Facility: http://www.tcf.vt.edu/
23. "Virginia Tech – System X Takes on the Grand Challenge": http://www.apple.com/science/profiles/vatech2/
24. Wikipedia – System X: http://en.wikipedia.org/wiki/System_X
25. COLSA Corporation "Taking Apple Xserve to MACH5": http://www.apple.com/science/profiles/colsa/
26. Detailed Notes from Virginia Tech Supercomputer Presentation: http://www.chaosmint.com/mac/vt-supercomputer/
27. Wikipedia – Fat tree: http://en.wikipedia.org/wiki/Fat_tree
28. Distributed-memory MIMD machines: http://www.top500.org/ORSC/2004/dm-mimd.html
29. "Computer Architecture, A Quantitative Approach" by John L. Hennessy and David A. Patterson, 3rd Edition
30. Wikipedia – Moor's Law: http://en.wikipedia.org/wiki/Moore%27s_Law
31. An InfiniBand Technology Overview: http://www.infinibandta.org/ibta/
32. Wikipedia – InfiniBand: http://en.wikipedia.org/wiki/Infiniband
33. An Introduction to the InfiniBand Architecture: http://www.oreillynet.com/pub/a/network/2002/02/04/windows.html
34. Wikipedia – HyperTransport: http://en.wikipedia.org/wiki/HyperTransport
35. Myricom/Myrinet: http://www.myri.com/
36. Quadrics: http://www.quadrics.com/
37. MPI for InfiniBand Project: http://nowlab.cse.ohio-state.edu/projects/mpi-iba/
38. OpenIB Developers Workshop Feb 2005: http://openib.org/openib_workshop_0205.html
39. Stanford SPIRE Parallel Interactive Rendering Engine: http://spire.stanford.edu/

Section 2 – Overview of System X

1. Virginia Tech Terascale Computing Facility. www.tcf.vt.edu/systemX.html
2. System X Faster, but Falls Behind http://www.wired.com/news/mac/0,2125,65476,00.html?tw=wn_tophead_2

3. Va. Tech speeds up Mac OS X supercomputer by almost 20% http://www.computerworld.com/hardwaretopics/hardware/story/0,10801,96933,00.html
4. Top 500. http://www.top500.org/

Section 3 – Dollars and Cents

1. Virginia Tech, System X Takes on the Grand Challenge - www.apple.com/science/profiles/vatech2/
2. Apple – http://www.apple.com/xserve/
3. Virginia Tech Terascale Computing Facility. http://www.tcf.vt.edu/systemX.html

Section 4 – System Cooling

1. Xserve G5 Technology Overview. http://images.apple.com/server/pdfs/20050912_Xserve_G5_TO.pdf
2. Virginia Tech Terascale Computing Facility. www.tcf.vt.edu/systemX.html
3. http://www.liebert.com/dynamic/displayproduct.asp?id=1077&cycles=60hz

Section 6 – Message Passing

1. A User's Guide to MPI. Peter S. Pacheco, Department of Mathematics, University of San Francisco.
2. Introduction to MPI Distributed Programming on Mac OS X developer.apple.com/hardware/hpc/mpionmacosx.html
3. The C Programming Language. Brian W. Kernighan and Dennis M. Ritchie.