Making A Beowulf Cluster

Using Sun computers, Solaris operating system and other commodity components

Peter Wurst and Christophe Dupré
Scientific Computation Research Center
Rensselaer Polytechnic Institute, Troy, NY, USA

1. INTRODUCTION:

The Scientific Computation Research Center (SCOREC) was established jointly by the Schools of Engineering and Science at Rensselaer Polytechnic Institute. SCOREC models real world physical reactions by simulating them on computers. The numerical solution of these models often requires discretizations with a number of unknowns approaching one billion. Problems of this scale can only be solved on large-scale parallel computers. For this reason SCOREC needed a multi-processor parallel computer of sufficient size to begin to address problems approaching this size.

One of the first approaches to running tasks on multiple processors was to create large multi-processor machines. There are two major types of these large multi-processor machines, MIMD (Multiple Instruction Multiple Data) and SIMD (Single Instruction Multiple Data) [1]. These machines, although very fast, are difficult to make, extremely expensive, and suffer from many problems, such as limited bandwidth from each processor to the memory. Because of their high cost an alternative for creating parallel machines was needed. One alternative method is the development of clusters.

A cluster is a collection of computers that cooperate in order to solve a large problem. In 1994 NASA began the Beowulf Project [2] at the Goddard Space Flight Center, later that year they produced the first operational Beowulf cluster. A Beowulf cluster [3] is a collection of commodity computers and communications hardware, running commodity operating system and software, capable of supporting MIMD parallel processing. The cluster that they presented was comprised of sixteen nodes and used 10BaseT Ethernet to communicate between the nodes. Soon after the presentation of the first Beowulf cluster [4] labs all over the world copied the concept.

High performance clusters (HPC) are another commonly found type of cluster. The major difference between an HPC and a Beowulf cluster is that when building a HPC, the performance of the entire cluster is given priority over low cost. Therefore, they are fast but also very expensive. According to the top500 list [5], the fastest current HPC is ASCI White at Lawrence Livermore National Laboratory [6]. The drawback of HPCs is that the cost per gigaflop is much higher than in a commodity cluster, but if you absolutely need the extra computational power, there is no other way to go. HPCs are also the only solution for problems involving large amounts of communication between nodes, due to the fact that commodity components aren't capable of handling large amounts of communications traffic between a great number of nodes in a scalable manner. The types of problems that are solved at SCOREC don't require an extreme

amount of communications between nodes, however, and for this reason we chose to build a Beowulf cluster.

The goal of our project was to create a scalable commodity cluster. At the same time, however, it was important to make sure that the communications system was not a major bottleneck such that a majority of a problems run-time was spent on communications. Since this cluster is to be part of a production research organization two additional goals of the project are that the cluster be easy to use and easy to maintain.

2. BUILDING THE CLUSTER:

The cluster was built using forty-eight Sun Netra X1s [7] as computational nodes, a Sun Enterprise 220R [8] as the centralized administration node, two Cisco Catalyst 3548XL Ethernet switches, two APW-Wrightline racks, and 6 APC Master-Switch Plus power distribution units. Each of the pieces was chosen for their performance, low cost, reliability, and maintainability.

The forty-eight Sun Netra X1s were excellent candidates for being made into a cluster. Sun's processors are known to handle a large number of floating point operations per second, which is very important in the scientific computations the cluster is going to be used for, and Sun hardware is known for its reliability. The X1 is 1U (the units measure for the height of a rack-mountable object, equal to 1.54 inches) in height and is made from commodity parts, including PC 133 SDRAM, 5400 RPM EIDE hard drives, and a 400MHz UltraSPARC IIe processor. The final configuration we used for the nodes is a 20GB EIDE hard drive and 640 MB of memory.

The Sun Enterprise 220R was chosen as the administrative front end for the cluster. The purpose of this computer is to run Sun's Solstice Autoclient server to boot all of the nodes from, so that they all have an identical configuration and are easier to maintain. It also serves as the master node for Sun's HPC cluster tools. At the writing of this paper there is a bug in the Autoclient software which prevents the X1 from being net-booted. The E220R was chosen for its reasonable cost and reliability features like redundant power supplies and mirror SCSI hard disks.

We chose the two Cisco Catalyst 3548XL because they each had forty-eight ports in a dense package (1U of vertical space), enough for all the nodes plus two GBIC slots for gigabit uplinks. One of the switches was used for the internal network dedicated to computational communications; the other was for the external network used for file sharing and user access. All of the nodes were connected to the external switch, which had a gigabit uplink to the rest of the network including the administrative node.

The APC MasterSwitch Plus, while not strictly necessary in order to get the cluster up and running, are important for ease of maintainability. The MasterSwitch Plus have web/SNMP management cards which allow remote administration via telnet and the web, and allow an administrator to remotely power on and off any connected machine

from the power source. The MasterSwitch also allows for a staggered power up, preventing the overload of the electrical circuits.

2.1 PROCESS OF CREATION

The first step in putting the cluster together was to record the MAC address of each of the nodes, so that they could be added to the jumpstart server. Each machine was then tested to make sure that it booted properly and didn't have any defects out of the box. This process is tedious, as it requires the person booting the machine to make sure that it passes all of the boot-time hardware tests. Every single one of the computers passed their initial boot tests.

Then the memory was upgraded and tested in each machine, in order to ensure that all of the machines worked before they were installed. This process was probably the most time consuming of all of the physical installation tasks. Each machine came with the standard 128MB of memory and then two 256MB DIMMs were added to each of them. At the end of the process, only two faulty memory chips were found, out of the ninety-six installed. Then all of the nodes, including the front-end node, the switches and the MasterSwitch Plus were installed in the racks. After this was done all of the machines were connected to both of the switches and the external network switch was connected, via gigabit uplink to the rest of the network. Then all of the cables and machines were properly labeled, thus ensuring that any number of nodes can be removed and the proper cables can be immediately found when it is re-installed or a replacement is put in its place. The most important of all of these steps, however, is to label the back of every computer as well as all of the cables attached to them. All of the machines should also be connected, in a logical way, to both the Ethernet switches and the Master Switch Pluses. The ports on the Cisco switches are labeled one through forty-eight, so connecting nodes one through forty-eight to their relative port on the switch, one through forty eight is the obvious choice. The MasterSwitch Plus are labeled one through eight on each, so connecting node one to Master Switch one port one is the logical choice and connecting node 9 to Master Switch two port one is also a logical choice. This makes administration of the ports on the Master Switch Pluses much easier, as you can easily figure out which port to turn off or on in order to power down or up and individual machine.

The Master Switch Pluses were then configured to accept both administrative web and telnet access and then configured to stagger power-on to each of the nodes. This step was perhaps the easiest of all of the steps, and the Internet administration of the switches is very self-explanatory. As a last step each node was jumpstarted[9], a method developed by Sun allowing machines to be installed and configured from a Jumpstart server on the network. This ensures that all of the nodes have the exact same configuration and cuts down on the amount of time it would take to install and maintain each machine individually. This step is very time consuming, because it requires that the administrator determine exactly which packages are going to be required in order to operate the machine. An excess of packages may cause the nodes to run slower, where as not having all the necessary packages may cause the machine to crash or prevent certain applications from being run.

3. SOFTWARE

Parallel processing requires special software in order for the processes on each of the nodes to communicate between each other. The most widely used software package for this purpose is MPI [10] (Message Passing Interface), and it is available from a number of vendors, including an open source version known as MPICH[11]. Sun also makes their own version of MPI which they distribute with the HPC Cluster Tools[12]. The benefit of using Sun's version was that it has been optimized to run on Sun's hardware. This was confirmed by running various programs on the cluster using both MPICH and Sun's MPI[13], and all of the programs ran faster using Sun's MPI.

3.1 MANAGEMENT SOFTWARE:

Any reasonably sized cluster would be far too difficult to use without specialized management software; users would be required to know how many nodes are available at any given time, who is running which program on which node, how to start a new program, etc. Special programs taking care of job queuing, scheduling and monitoring are available. These programs can be used on non-cluster and cluster-type machines, but they are really necessary to make efficient use of clusters in general. A number of commercial and Open Source programs cluster management software [14] are available to take up the task. Among others are Condor [15], DQS [16], NQS [17], LSF [18] and Sun Grid Engine (SGE) [19]. Such programs generally offer a queuing software that allows users to submit new batch jobs to run on the cluster, a scheduler that organizes in which order the queued jobs will be run and on which nodes, and a monitoring software that allows one to see running jobs and also to stop them. Many software also have advanced functionality, like job migration, job check pointing/restarting, heterogeneous cluster support, multiple job priority, etc. Each software has its own way of doing things, and software will often need to be modified to make use of these advanced capabilities.

Since our projected cluster was an homogeneous cluster of dedicated machines, we were happy with a management system that offered basic functionality. The other things we were looking for are support for MPI, support for multiple network interfaces and a nice user interface. We chose Sun's Grid Engine solution because it was free, is supported by Sun and had the features we were looking for.

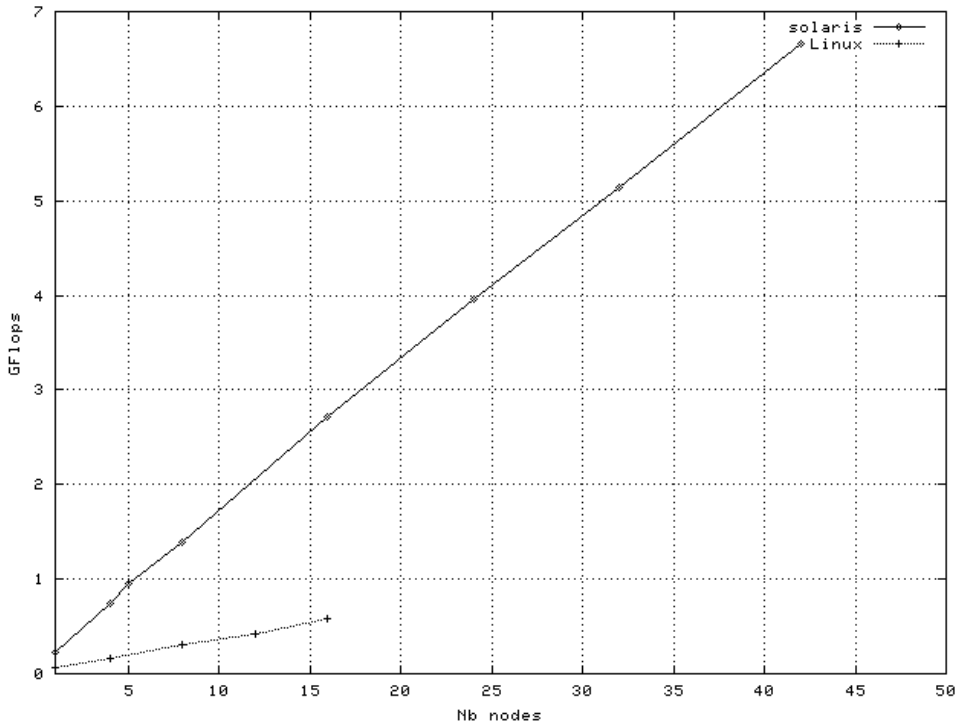
Sun's Grid Engine was previously known as Codine, which was based on the University of Florida's DQS. The system offers a great deal of flexibility, and is designed to work both on dedicated computing machines and on network of workstations in a 'cycle stealing' mode. The system works by defining 'resources' and 'queues', and specifying what resources are available for each queue (memory, number of processors, type of jobs that can be run, etc). When users submit jobs, they can specify resources required to run the jobs. The scheduler will then match job requirements with queue specification, as well as any access list configured, to determine if a job can be run and where. SGE also comes with scripts to integrate MPI into the system: when a job is scheduled, scripts are run to prepare the environment for MPI. Other scripts can be run afterward to clean-up.

Using SGE, users need only to determine the resources required to run their job (number of processes, memory per process, licenses required, etc). The management system will take care of the scheduling. This enables the administration team to remove and add nodes dynamically, without the need of information the users of the changes. SGE will make maximum use of the available computing resources.

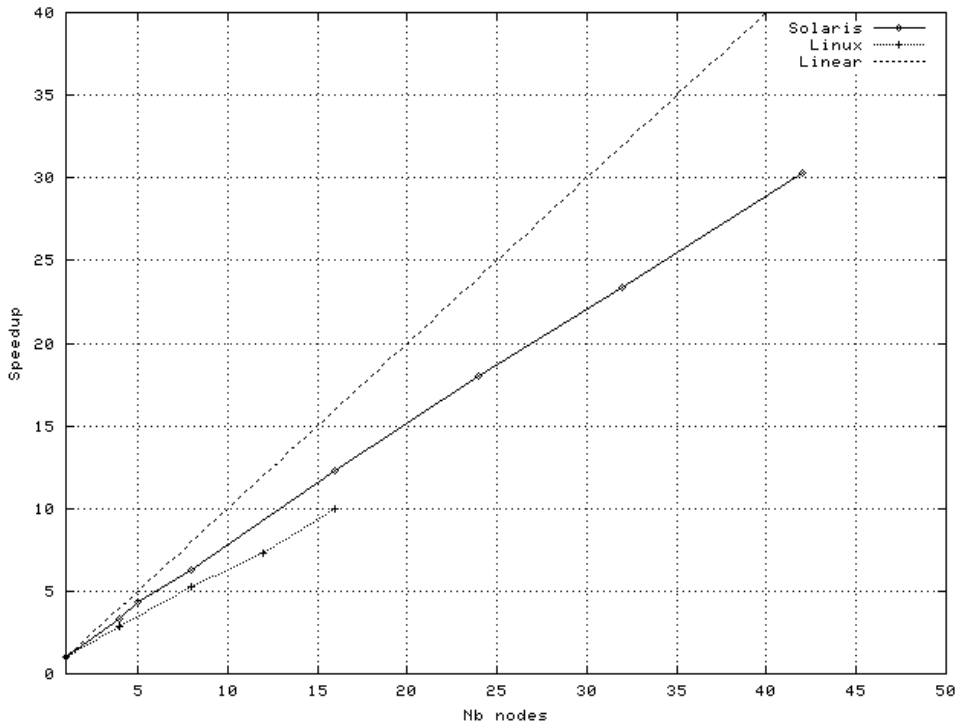
4. PERFORMANCE RESULTS:

In order to test the performance of the cluster we chose two main benchmarks: Linpack [20], to test the floating-point computational performance of the cluster, and bisect [21] to test the communication bandwidth between the nodes. Linpack is a well know cluster benchmark tool and is used by the top500 list in order to rank the most powerful computers in the world. It consists of a group of Fortran subroutines that analyze and solve systems of linear equations as well as least square estimates. It also computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems. Linpack stresses the floating point processor (fpp) to its limits and gives very accurate results on how well the fpp will perform in scientific applications. Bisect is a simple program that sends packets of data from all nodes to all nodes, and times the length of the transfers in order to give an average MB/s bandwidth figure per node.

Linpack runs on the Sun cluster gave very good results and showed a high level of scalability. Individual nodes are capable of 195 MFlops and on tests on up to 42 nodes the cluster realized nearly an eighty-two percent scalability factor a speedup of more than 30, with each node contributing 159 MFlops, for a total of 6.666 GFlops. A similar set of runs were done on the Linux cluster [22] jointly owned by SCOREC and the Computer Science department at RPI, comprised of four quad-processor PIII Xeon 550 MHz machines. Each processor on these machines was capable of nearly 51 MFlops, but had showed poor scalability of about 71 percent with a speedup of barely 10 for 16 processors. The main probable reason for the poor scalability of the Linux cluster was that there was a lack of memory bus bandwidth from the four processors across the bus to the memory in each node. Each node on the Linux cluster had a performance of about 36 MFlops when using the full sixteen processors.



Graph 1: This graph shows a comparison between the number of GFlops realized from both the Sun cluster and a Linux cluster based on Linpack runs.

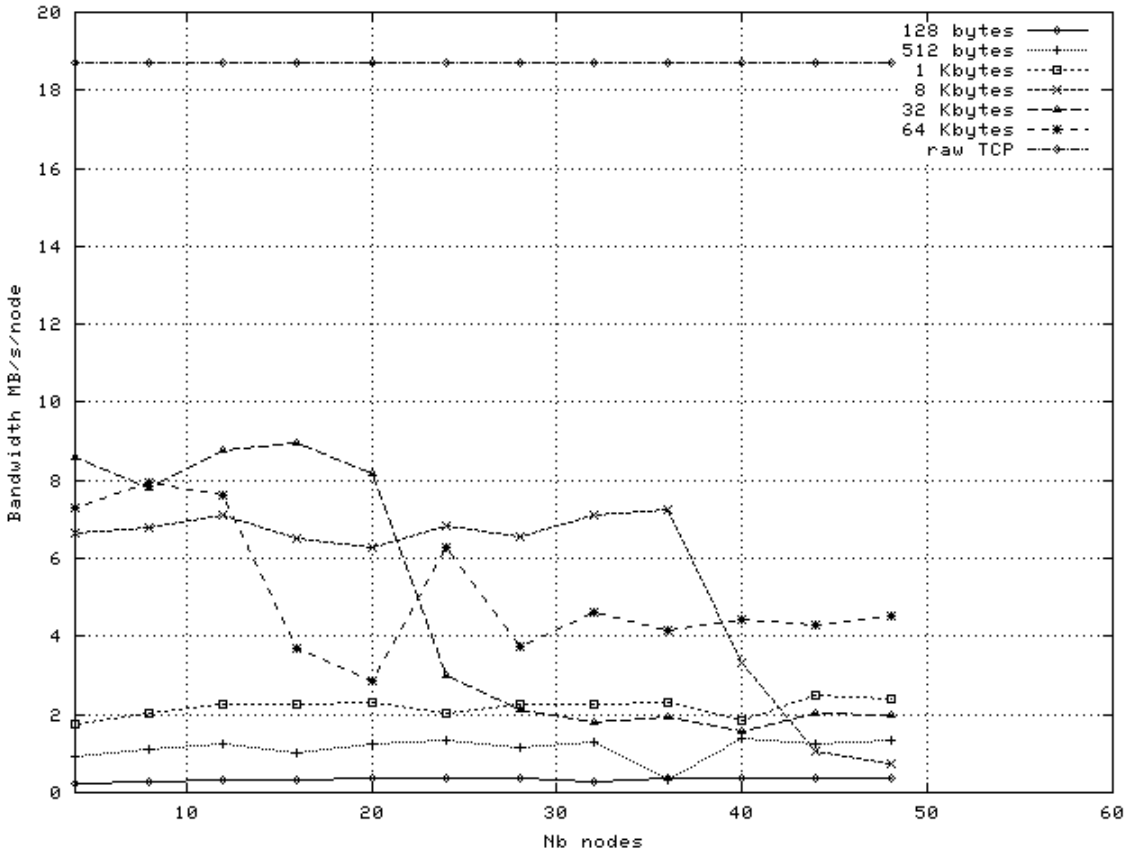


Graph 2: Comparison of the speedup due to an increase observed when increasing in nodes on both the Sun cluster and Linux cluster for the same Linpack runs, and the ideal linear speedup as Graph 1.

The bisection bandwidth is "the number of data paths that cross the bisection (an imaginary surface that divides a parallel computer into two equal halves) times the data rate in each data path." [1]. In this cluster's case, all the nodes are connected to a central switch with a simple backplane, giving a standard star topology. Thus given *n* the number of nodes, and *Bn* the average bandwidth per node, the bisection bandwidth *B* can be computed as:

$$B = \frac{n \cdot Bn}{2}$$

Bisect [21] tests on the sun cluster, which experimentally determined the effective average bandwidth per node of the cluster, gave results based on the scenario that all of the nodes are communicating at the exact same time. These tests gave great results for communication between up to 36 nodes, with average bandwidth on each node staying relatively constant. Above 36 nodes bandwidth began to drop between nodes as the switch became overloaded. However, this is a worst-case scenario and it shouldn't impact scientific computations very much as in real scientific computations the nodes spend much time computing and relatively little time communicating.



Graph 3: Average bandwidth per node for a packet size of 128 bytes to 64KB and raw TCP performance.

As a basis for comparison, graph 3 also includes the raw TCP bandwidth between nodes as determined experimentally by netperf [23] This bandwidth is the wire speed (100 Mbps) minus the various protocol overhead (Ethernet, IP, TCP, operating system). The

difference between the raw TCP and bisect numbers can be explained by the overhead incurred by MPI synchronization.

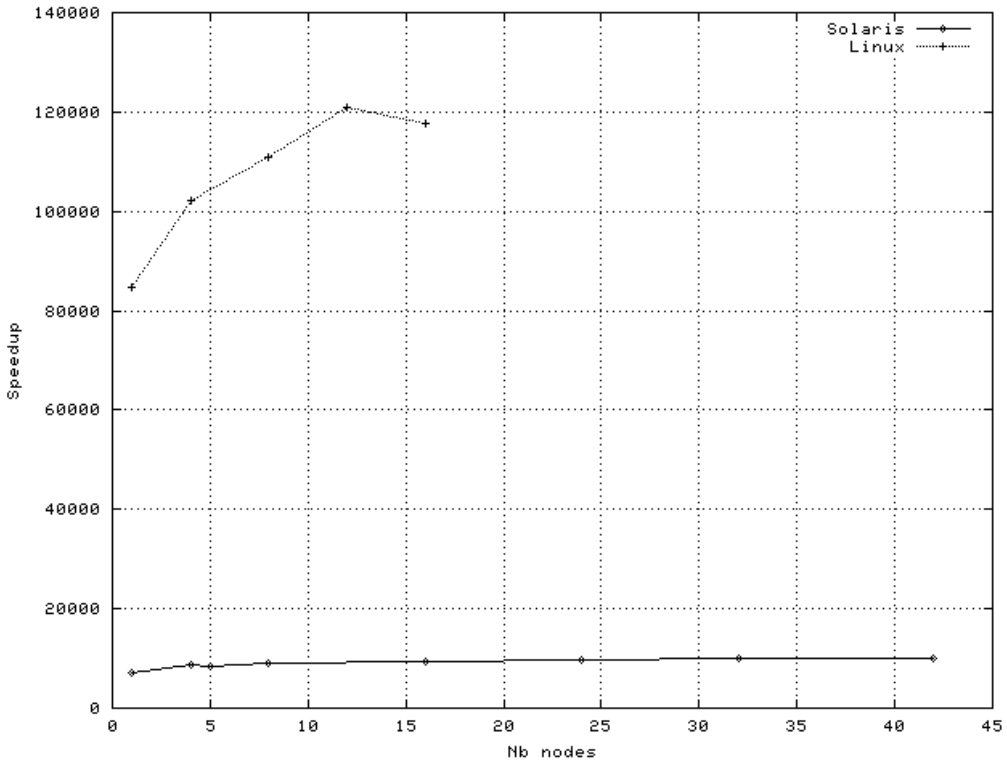
Initial cluster test for actual SCOREC scientific computations has begun. The first tests were fluid instability analyses using an adaptive discontinuous Galerkin discretization of several million unknowns subjected to several thousand explicit time steps. In addition to the interprocessor communications required for the standard gather/scatter operations needed in matrix operations, adaptive simulations require interprocessor communications to support dynamic load balancing as the spatial mesh is adapted every hundreds of times during the simulation. Initial results obtained indicate the cluster is performing well even on this demanding application. Additional study and results will be forthcoming shortly.

5. COMMODITY SYSTEM COST PERFORMANCE MEASURE

A commodity cluster is made from commodity parts in order to get a high performance for a low amount of money. In order to judge if the Sun cluster has met this goal, we tallied up the cost of the entire cluster and then divided the cost between all of the nodes to determine the cost per node. The cluster's cost per GFlop is given by:

$$C_p = \frac{n \times C}{P_n}$$

where n is the number of nodes, Cp is the cost per gigaflop using n nodes, C is the cost per node to build the cluster, and Pn is the performance measured using n nodes. Graph 4 shows the above formula computed for both the Linux cluster and the Sun cluster.



Graph 4: Cost per gigaflop of each of the clusters.

Our Sun cluster performed a lot better in cost per gigaflop, about fifteen times cheaper than the Linux cluster on the limited tests done to date. This answered the question as to whether or not our cluster was considered a commodity cluster or not, seeing as the Linux cluster is almost the text book definition of a commodity cluster. It also means that our cluster is indeed a Beowulf cluster.

Note that the ratio of 15 obtained was for a single test run on a Linux cluster that was constructed two years ago and that is not an optimally configured. Therefore, one should expect that ratio to be lower if done on a Linux cluster built today. Although we do not know that exact ratio, it is clear that the Sun cluster is a strong candidate for the construction of Beowulf clusters.

6. CONCLUSION

The Sun Netra X1 is an ideal computer for making into a commodity cluster of a limited number of nodes (estimated to be less than or equal to sixty-four when a reasonable level of interprocessor communications are required). The Sun microprocessor is faster for floating point operations than similarly priced Intel processors, so they are ideal for scientific computing. This bundled with the fact that there is a large amount of software available for administering and using Sun computers as well clustering them makes them the perfect solution for large computing needs in scientific environments.

7. REFERENCES

[1] R. Michael Hord. Understanding Parallel Supercomputing, IEEE Press, 1999

[2] Center of Excellence in Space Data and Information Sciences (CESDIS), NASA Goddard Space Flight Center. http://www.beowulf.org/

[3] Donald J. Becker et al. BEOWULF: A Parallel Workstation for Scientific Computation, Proc. Int. Conf. on Parallel Processing, Aug 1995, Vol 1, 11-14

[4] Daniel Ridge et al. Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs, Proc. IEEE. Aerospace, 1997

[5] TOP 500 supercomputers Sites. http://www.top500.org

[6] Using ASCI White. http://www.llnl.gov/asci/platforms/white

[7] Sun Netra X1 rack-optimized server. http://www.sun.com/products-n-solutions/hw/networking/netrax/X1/

[8] Sun Enterprise 220R workgroup server http://www.sun.com/servers/workgroup/220r/index.html

[9] Kasper Paul Anthony and Alan L. McClellan. Automating Solaris™ Installations:A Custom JumpStart Guide, Prentice Hall PTR, 1995.

[10] Argonne National Laboratory. The Message Passing Interface (MPI) Standard: http://www.mcs.anl.gov/mpi/

[11] MPICH homepage. http://www-unix.mcs.anl.gov/mpi/mpich/

[12] SUN HPC Software Homepage. http://www.sun.com/software/hpc/

[13] SUN HPC Overview and MPI. http://www.sun.com/software/hpc/overview.html

[14] Mark A. Baker, Geoffrey C. Fox and Hon W. Yau
Cluster Management Software, The NHSE Review, 1996 Volume, First Issue, http://nhse.cs.rice.edu/NHSEreview/CMS/

[15] Condor High Throughput Computing project. http://www.cs.wisc.edu/condor/

[16] DQS Distributed Queuing System. http://www.scri.fsu.edu/~pasko/dqs.html

[17] Network Queueing System. http://umbc7.umbc.edu/nqs/nqsmain.html

[18] Load Sharing Facility. http://wwwinfo.cern.ch/pdp/lsf/

[19] SGE - Sun Grid Engine. http://www.sun.com/gridware

[20] Linpack benchmark. http://www.netlib.org/linpack

[21] The cluster benchmark web page. http://www.di.unito.it/~mino/cluster/benchmarks/

[22] Rensselaer High Performance Computing Cluster. http://cube0.cs.rpi.edu

[23] Netperf benchmark tool. http://www.netperf.org