ClusterFLOP

An Experiment in Building a **Cluster** **F**rom **L**ow c**O**st **P**arts

E.E. 505 - Computer Architecture – Professor Swarz
E.E. 506 - Introduction to LANs and WANs – Professor Gannon

David Feinzeig
david@wpi.edu

Jeremy Slater
jasl8r@wpi.edu

May 3, 2004

Acknowledgements

We would like to extend special thanks to the following parties:

- Professor Donald R. Brown for allowing us to use space in the **spin**lab.
- Stephen W. Laverty and Benjamin Woodacre, WPI, class of 2003, for their invaluable suggestions and assistance.

Contents

List of Figures

1 Introduction

The classical approach to improving computing performance has been to "make it faster". Competing manufacturers have played a game of oneupsmanship, reducing process sizes and increasing processor operating frequencies. This ideology allows consumers to purchase the latest and greatest processing power at a premium. Large-scale parallel computing systems have typically been constructed from such high-end components. However, recently there has been a branch away from using the best (and most expensive) components available towards using lower cost, mid-grade parts. This new concept of parallel computing has become more about using readily available resources to perform the same operations at a reduced cost.

The introduction of Local Area Network (LAN) switches has resulted in the ability to create scalable massively parallel computing systems with high network bandwidth from off-the-shelf components, instead of having to rely on custom systems and custom networks. The use of standard parts also allows for a much less expensive implementation of a parallel computer. Since a cluster consists of separate computers connected through a networking infrastructure, replacing or repairing a node does not mean that the entire cluster has to be taken offline. Repair may be as easy as unplugging the defective node, without affecting the operation of the rest of the cluster, and connecting a new node.

One disadvantage to running a cluster has been the absolute cost of owning one. Managing a cluster of N nodes can be close to the cost of managing N separate computers. However, a better measurement is cost for performance. A cluster can have the same performance as a large-scale proprietary parallel computer, but can cost less due to the use of highly available parts. Large-scale proprietary parallel computing systems are produced in small volumes using specialty components, resulting in higher development fees being incurred per system. A cluster is composed of commodity parts, produced in large volumes for general markets and not specifically supercomputers, resulting in much smaller development costs being passed onto the purchaser.

This project was an experiment in constructing a cluster from low cost parts. A 2 GFLOPS system of 10 nodes, with 1.6 GBytes of total RAM and 10 GBytes of total hard drive storage was built on a 100 Mbps Ethernet network at a cost to the authors of approximately $1,000. A head node with a single hard disk is used to boot the remaining 9 diskless client nodes over the network. OpenMosix is used on a Debian Linux operating system to provide an easy to use and manage single system image clustering environment. The final ClusterFLOP is meant to be a proof of concept with results that are scalable to compete with larger-scale parallel computing systems.

2 System Architecture

ClusterFLOP is composed of 10 nodes (1 master and 9 clients). The master node has 512 MBytes of RAM and 10 GBytes of hard drive storage. Each client node has 128 MBytes of RAM and no hard drive storage. Each node is connected via an onboard 100 Mbps Network Interface Card (NIC) to a non-blocking 100 Mbps switch. The master node has an additional PCI NIC that serves as the Wide Area Network (WAN) connection. The head node is used to boot the 9 client nodes over the network through the use of RAM disks and network file system protocols.

Debian Linux is used in conjunction with the OpenMosix kernel extension to provide a single system image (SSI) clustering environment. The master node has a complete installation of Debian Linux with a kernel that has been patched with OpenMosix. Each client node is identically configured to run an optimized OpenMosix-enhanced Linux kernel in RAM and has read-write permissions to a hard drive location on the head node. All interaction with the cluster (e.g. - running applications, cluster status checks, etc.) is accomplished through the head node. There is no need to log onto the client nodes due to OpenMosix's ability to transparently migrate processes in order to distribute the process load to all nodes.

The chain of events that occurs from the time that the power switch is turned on to when the cluster is operational is as follows:

1. The power is turned on and the master node begins to boot Debian Linux as the client nodes start the network boot process, pausing as they search for a Dynamic Host Configuration Protocol (DHCP) server.

2. Once the head node's operating system has loaded, it starts a DHCP server and assigns each client node an Internet Protocol (IP) address. At this point, each client node can now access a Trivial File Transfer Protocol (TFTP) server located on the head node, which uses the Etherboot service to send the kernel image to the requesting client.

3. Now each client node begins to boot using the optimized OpenMosix-enhanced Linux kernel. During the boot process, each node mounts a common root (/) directory (as read-only) on the head node via the Network File System (NFS).

4. Next, each client requests a mount location for */etc*, */var*, */tmp*, and */mfs* (OpenMosix File System), which must be mounted with read-write permissions. The head node responds to these requests by generating a unique directory structure containing copies of the four previously mentioned mount points for each client, based upon their IP.

5. Now that each node has a complete file system mounted they complete the boot process normally. Once the client has finished booting, the OpenMosix extension is enabled, allowing the node to be added to the OpenMosix map, which identifies the available pool of nodes.

OpenMosix has a robust auto-discovery daemon, such that every time a client node boots in the above fashion it joins the OpenMosix cluster. This allows the master node to re-balance the overall cluster process load by migrating processes to the new node. The daemon also recognizes if a node should suddenly become disconnected from the cluster and adjusts the overall cluster process load appropriately.

The hardware and software systems that permit the above-described cluster operation are explained in more detail in the following sections. Testing techniques and results are discussed as well, followed by conclusions obtained from the completion of this project.

3 Hardware System Design

ClusterFLOP relies upon several different sets of hardware specifications. This section discusses the specifications for the master and client nodes, how the 10 nodes were powered, the network configuration used, and the methodology used to accomplish the diskless network booting of the client nodes.

3.1 Node Specifications

Each node is running a Syntax S635MP motherboard with an integrated VIA C3 800 MHz Samuel2 processor. Unfortunately, after having purchased the motherboards, when reviewing the datasheet for the processor it was discovered that, "[t]he FP unit is clocked at 1/2 the processor clock speed; that is, in an 800-MHz VIA C3, the FP-unit runs at 400 MHz." [1] To make matters worse, "[o]ne floating-point instruction can issue form [*sic*] the FP queue to the FP unit every two clocks." [1] This means that the Floating Point Unit (FPU) is effectively running at 1/4 the speed of the processor, or 200 MHz. Based upon the original impression that the FPU ran at the full speed of the processor (800 MFLOPS for each of the 10 processors), the intention was to achieve a maximum performance of approximately 8 GFLOPS across the entire cluster. However, the uncovering of the true FPU speed as being only 200 MFLOPS for each of the 10 processors means that the maximum attainable performance is actually 2 GFLOPS for the entire cluster. This factor of 4 discrepancy was disappointing, and although it affected the absolute performance of the cluster, it was still possible to demonstrate the proof of concept for the construction and relative performance of the system.

Every node is connected to the Linksys 4116, 16-port, 10/100 Mbps, non-blocking switch via an onboard 10BASE-T/100BASE-TX NIC, which is composed of a SiS900 chipset with a Realtek RTL8201 LAN PHY transceiver. In addition, the master node has a PCI Linksys NC100 NIC, which acts as the WAN connection for the cluster.

The only hard drive storage for the cluster is a 10 GByte disk on the head node. This node also has 512 MBytes of RAM and each client node has 128 MBytes of RAM.

(Due to unexplained intermittent problems with the hardware of node 10, this client node has been taken offline to provide more accurate testing of ClusterFLOP.)

3.2 Powering of ClusterFLOP

For monetary, AC to DC power conversion efficiency, and aesthetic reasons, it was decided that the 2 groups of 5 nodes would share power supplies. The amount of power consumed by a single board was an important measurement that played a role in deciding what minimum rating would be necessary on purchased power supplies. This was measured by tying the multiple 3.3 Volt, 5 Volt, and 12 Volt power supply connectors together, respectively, and measuring the amount of current drawn by a single node at each voltage with an ammeter. The resulting power draw is approximately 30 Watts per node. Therefore, each power supply needs to be able to supply about

150 Watts to its group of 5 nodes. The actual power supplies that were used are 330 Watts apiece, which is more than adequate to power the nodes.

An important concern was how to split up each of the two 330 Watt power supplies into 5 ATX power connectors that would plug into each node's motherboard. This issue was resolved by designing a simple printed circuit board with ATX power connectors soldered onto it that acted as a 1-input to 5-output ATX power supply bus. Two of these boards were fabricated by Advanced Circuits and then 10 male and 2 female ATX power connectors were attached to the buses. This proved to be a satisfactory solution to the problem of powering the 10 nodes.

3.3 Diskless Network Booting of Client Nodes

It was decided that the head node would be used to boot the 9 diskless client nodes over the network. In order for this to function properly, the onboard NIC driver integrated into the BIOS had to support DHCP/TFTP via IP. However, once powered on and configured in the BIOS to boot from the network, it was discovered that the BIOS driver was in fact compatible for use with Novell NetWare via IPX only. Since the motherboard manufacturer provides no alternate NIC driver, this meant that we had to replace the driver module located in the BIOS with the appropriate one. This process required that we properly identify the exact NIC device implemented on the motherboard, generate an appropriate ROM file to substitute for the incorrect driver, substitute this new driver for the correct BIOS module, flash the BIOSes, and keep our fingers crossed.

The information available on Syntax's website suggested that the NICs used were SiS900 devices. This was confirmed by examining the vendor and device identifiers located in the BIOS module for the device. The next step of generating the required NIC driver was accomplished through the use of EtherBoot. "Etherboot is a software package for creating ROM images that can download code over an Ethernet network to be executed on an x86 computer." [2] This Linux-based network bootloading software was obtained from ROM-o-matic.net [3] and, after many failed attempts, required some source code adjustments due to an erroneous lookup table in the C source file responsible for identifying devices that belong to the SiS900 class of NICs. Once the problem was tracked down and fixed, and the source was compiled, the EtherBoot software was able to generate the correct ROM image that was required to netboot using PFTP/TFTP via IP. Next, it was necessary to extract a copy of the original BIOS from a motherboard and replace the original NIC driver module with the new one.

A BIOS specific manipulation tool was necessary to complete the rest of this task. Internet postings suggested that since the motherboards use an American Megatrends BIOS (AMI) [4], the AMIBios Configuration Utility (AMIBCP) was the appropriate choice. Since this utility allows for editing of the BIOS, including unlocking of secured motherboard manufacturer determined settings and menus as well as access to the onboard driver modules, AMI has attempted to keep it under 'lock and key' by limiting distribution to corporate purchasers of their BIOS. Needless to say, it was extremely difficult to find, although not impossible. After much searching on the Internet, AMIBCP[.exe] version 7.51.03 was located and installed. [5] This allowed us to substitute the newly generated NIC driver module for the original. Once inserted into the BIOS, one of the motherboards was flashed with the modified BIOS using the AMIFLASH Flash EPROM Programming Utility version 8.27.38 provided by AMI. [4] The motherboard was then powered up as we watched with

fingers crossed. The BIOS was able to correctly locate and load the driver for the SiS900 NIC and presented us with the message "Searching for DHCP Server...". This meant that the motherboard was now ready to locate a DHCP server, request a kernel image, and boot from that image. At last, the 10+ hours of work required to overcome this unexpected obstacle proved to be successful.

4 Software System Design

With the required hardware infrastructure in place, it was necessary to run the appropriate software. The master node's operating system is a standard Debian Linux installation. The OpenMosix kernel extension is patched into the kernel. Each client node runs an identical kernel with a custom initrd. Specialized scripts are used to properly boot and configure the nodes of the cluster. OpenMosix provides certain "Userspace tools" to simplify the use and monitoring of the cluster. These software aspects of the cluster are further explained in this section.

4.1 Etherboot Network Boot Image

The client transfers an Etherboot network boot image (*etherboot.nbi*) from the boot server on the master node via TFTP. This file contains a kernel image and optionally an initial ram disk (initrd) if the kernel was built to use one. In our case, the initrd is necessitated by a combination of an OpenMosix requirement, namely that the same kernel image run on all nodes, and our need to have the head node boot from its hard drive and all other nodes to netboot. The initrd also allows for easy reconfiguration of the booting process which was valuable for developing our process. These two files are packaged into a compound file of the appropriate format via the *mknbi-Linux* utility.

While the standard initrd image is appropriate for the head node, as it performs a standard boot, a custom initrd must be assembled for the other nodes. The kernel modules, utility binaries, and scripts included in the initrd can be customized using the *mkinitrd* utility. The netbooting initrd we assembled includes a lightweight DHCP client and custom startup scripts. The normal boot sequence involves mounting a root partition located on the master node's hard disk and then pivoting that mount to root (/) in place of initrd (which is mounted as root up to that point). In the netboot initrd startup script we first obtain our IP address using the DHCP client and configure the Ethernet adapter (even though the new NIC boot ROM – refer to Section 3.3 – obtained an IP and had the Ethernet adapter functioning earlier in the boot process that information is no longer available once we start the kernel). Next we mount an NFS partition from the host node and pivot that mount to root in place of the initrd and continue booting.

4.2 OpenMosix Required Client Node Configuration

The key to having a cluster of diskless machines all operating safely is mounting the correct partitions with the correct file permissions. Each node needs write access to four different directories: */etc*, */var*, */tmp*, and */mfs*. The rest of the disk can be mounted as read-only. This means that there are only four directories that need to be unique between nodes, the rest of root (/) can

be shared by any number of devices, because they will only read the data from these locations. This greatly reduces the required disk space in a multinode system, because each node only needs, in our case, 36 MBytes of unique storage. If the entire drive had to be unique, then the storage requirements would be on the order of 300 to 500 MBytes.

During boot, each node mounts the root file system as read only. At this point, if they have never connected to the server before, then there is no unique storage for the node, and it must be created by the server. We found an existing script and software combination online which proved to work fairly well, provided by Markus Amersdorfer [6]. With this system, the server or head node, runs software to listen for the clients. During the client startup they run a script that requests a mount point for the four unique directories. When the server gets this request it copies */etc* and */var* from their original locations to a directory specified by the clients IP. Both */tmp* and */mfs* are created empty, since they are empty on startup. When the server finishes generating these directories, the client resumes booting by creating a ramdisk locally to map to the newly created unique directories. When the clients finish booting, they have complete access to the writable directories and join the OpenMosix cluster.

4.3 OpenMosix Userspace Tools

The OpenMosix collection includes a suite of userspace tools. These utilities are designed to inform the user of the cluster status, as well as provide powerful control over the system. To begin with, there are such tools as *mosmon*, *openmosixview*, and *openmosixmigmon*. We primarily used the *mosmon* utility for its simple yet informative interface. This utility shows the cluster nodes along the bottom of the screen and the relative load is displayed as a bar graph above each node. Figure 1 shows this application while running a parallel execution of POV-Ray. This utility is helpful in showing which nodes in the system are active, and how well the processes are migrating. For instance, if we only start seven processes on a nine node system, then we will notice those seven processes migrate, leaving two machines idle. Both *openmosixview* and *openmosixmigmon* are graphical tools for use under X11. The *openmosixview* utility is a graphical version of *mosmon* which also includes information about each node's memory, the total system migration efficiency, and allows the user to alter the system's perceived CPU power. The *openmosixmigmon* software provides information about which processes are migrated and where they are migrated.

Additionally, OpenMosix contains some important control utilities such as *mosctl* and *mosrun*. The *mosctl* utility allows a user to specify characteristics about the OpenMosix cluster, or even tell OpenMosix to perform tasks with currently running applications. In particular, using the *mosctl* utility, the perceived speed of an individual node can be decremented. By reducing the speed value, the computer will appear as being slower, and OpenMosix will prefer to migrate processes away from this machine. This is useful in a system, such as ours, where there is a single head node with a GUI that needs to be responsive for a user. The *mosrun* utility offers users methods for controlling the migration of processes. At execution time a user can tell a process specifically not to migrate to other nodes, or which nodes it can migrate to, or even to only execute on one specific client node. There are many other complex command line options for this utility as well as many other OpenMosix programs, but they are beyond the scope of this project.

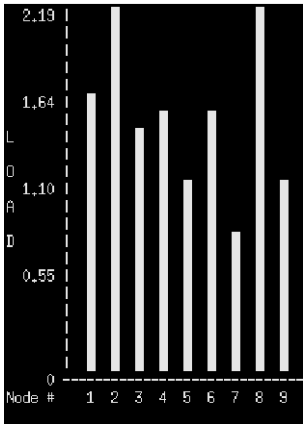

Figure 1: OpenMosix *mosmon* Node Monitoring Utility

5 Design for Testability

An important aspect in a high performance parallel computing environment is the speed of the network infrastructure and the level of network saturation. Ensuring that the network is not over utilized is especially critical in a diskless parallel computing environment such as ours, where the file system access and process management occur over the same interfaces. The level of utilization will vary greatly depending on the level of I/O or the size of data files used by migrating processes. If the network becomes too saturated with traffic, then the speed of the entire cluster will drop as I/O bound processes stall.

5.1 Monitoring Network Activity

5.1.1 Transparent Network Bridge

A tool to examine the network load between the head node and the switch is a transparent bridge configured on a PC running Debian Linux. The bridging functionality is built into kernel versions 2.4 and higher. Since a bridge forwards at the second layer, it is fast, but still allows an application to capture frames for later analysis. To ensure that the bridge was not introducing too much latency, we conducted tests using the ping utility to measure trip time between two machines. As our base comparison, we connected two computers directly with a crossover cable. We then connected them over the switch, so that we could measure the average introduced latency of the switch. Finally, we connected the host to the bridge which in turn connects to the switch and on to the client. For each network path the ping command was repeated 500 times and averaged, with the results shown in Table 1. It is safe to assume that our network will be relatively unaffected by the 0.056ms added delay from the bridge.

We used Ethereal [7] to capture all the network traffic between the head node and the switch for the duration of each test. This resulted in capturing every packet that passed through the bridge into a file. We were not concerned with the data in the packets, but rather just the packet statistics. By running tcp-dump on this file, we obtained a text file containing the basic information such as

Path	Delay	Δ Delay
Host → Client	0.137ms	0ms
Host → Switch → Client	0.154ms	0.017ms
Host → Bridge → Switch → Client	0.210ms	0.056ms

Table 1: PC to PC Delays

time of day, source, destination, packet type and packet length for every packet. Using Matlab, we parsed and stored the network traffic characteristics indexed by time so that we could graph the transfer rate over time in units of bytes per second.

5.1.2 Data Transfer Rate Analysis: MFS and NFS

The first tests conducted measure the maximum achievable transfer rates of files copied over the network using both MFS and NFS. We performed this test only between two nodes (the head node and a client) in order to judge the upper bound of each protocol. By discovering these upper bounds, we can tell if the network is saturated during our parallel execution experiments. The testing consisted of simply copying a 100MB file from the server to a node. As the file was copied, we captured the traffic and graphed the data transfer rate in Matlab. Figure 2 shows the results from the NFS test while Figure 3 provides MFS results. From these figures, it is obvious that the NFS protocol allows for a much higher sustained data transfer rate of nearly 100Mbps, as opposed to the MFS which yields approximately 60Mbps. This difference is easily explained; NFS transfers data over the network using UDP, a connectionless protocol, while MFS transfers data with TCP, a connection oriented protocol. There are additional overheads in TCP not present in UDP, such as opening and closing connections and error control. In fact, the OpenMosix How To [8], specifically says that MFS implements caching, time stamping and link consistency, all of which are not present in NFS.

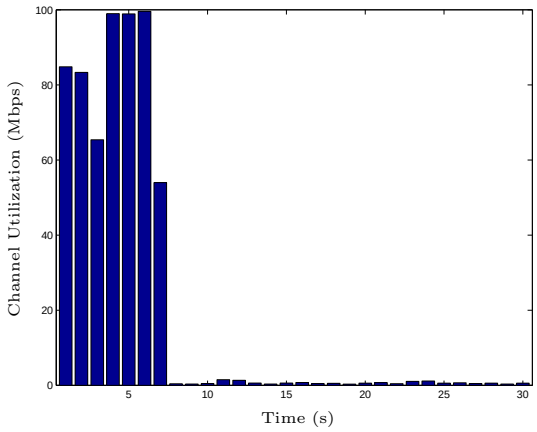

Figure 2: Maximum Transfer Rate of NFS

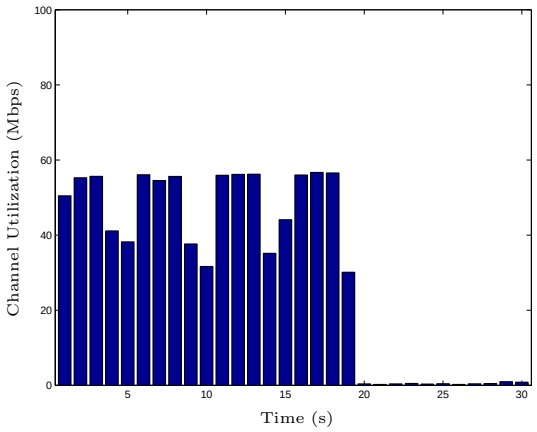

Figure 3: Maximum Transfer Rate of MFS

5.1.3 OpenMosix Node Monitoring Feature

An additional feature of the OpenMosix protocol is the heartbeat detection mechanism. During idle spans, the UDP traffic peaks every few seconds as it monitors the status of other nodes currently in the network. Figure 4 shows this heartbeat pattern over a two minute idle period. This pattern is shown clearly by hiding, in Matlab, the NFS traffic present at the same time.

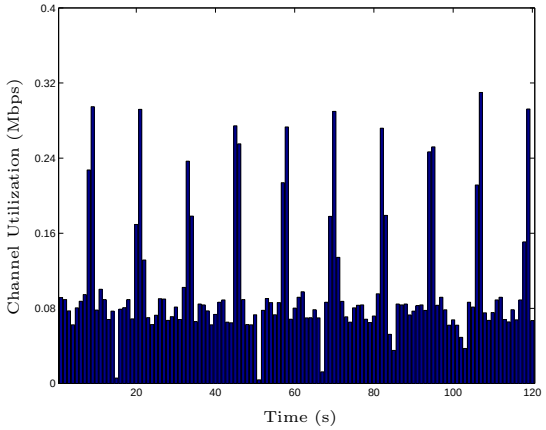

Figure 4: OpenMosix Heartbeat Traffic

5.2 Testing Sample Application: POV-Ray

OpenMosix is best utilized in an environment with highly parallelizable CPU intensive applications. A field of applications requiring a large amount of CPU power is 3d rendering, whether it is rendering single images or entire movies. These complex renderings can easily take days on a single system. However, this time can be reduced when rendering in parallel on a cluster. We are using POV-Ray (Persistence of Vision Raytracer) [9] on our cluster to render a simple test image. POV-Ray lets the user define a start row and end row when rendering, so the image can be split into at least as many processes as there are nodes in the system. Thus, the entire cluster can participate in the rendering. We designed a script, ppovray, which takes the number of image pieces as its first parameter, followed by the normal POV-Ray command line options. This script determines what size to make the chunks based on the total height specified for the overall image. It then executes POV-Ray passing along the dynamically created start and end line arguments. When every instance finishes, the script stitches all the partial images together into the final output scene. In testing, we have found that it is best to specify at least twice as many pieces as there are nodes in the cluster. This way, for images that are not evenly difficult to render, machines that finish processing easy sections will migrate another rendering process from a machine that is processing a harder section. In the ideal case, all nodes in the cluster will finish rendering at the same time.

We used the ppovray script in conjunction with the time command to perform multiple tests with varying numbers of connected nodes. We rendered a simple repetitive tile image, thus minimizing the range of processing difficulty amongst the partial renders. The final image has a resolution of

1024 by 768 pixels with 8X antialiasing. We started testing with all nine nodes connected and one by one, removed nodes from the system, in each case performing the test three times and averaging the results. This way, we ended up with a mean render time for each of one through nine connected nodes. In addition, as a comparison, we rendered the same image three times on an AMD Athlon XP 2800+ (2.08GHz) based computer running the same Debian Linux kernel. The results of this test are shown in Figure 5.

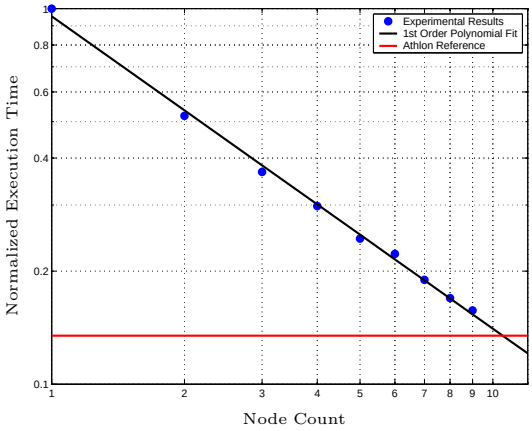

Figure 5: Normalized Execution Time of POV-Ray vs. Node Count

The resulting execution times shown in Figure 5 as blue dots are normalized to the length of time that a single node takes to render the image (e.g. - rendering an image using four nodes takes roughly 30% of the time that it takes a single node to render the same image). The relationship between execution time and the number of nodes is exponentially decreasing and so the data is plotted logarithmically so that it appears roughly linear. As a reference, the relative completion time for the Athlon is shown as a red bar. Using the polyfit function in Matlab, we generated a first order polynomial equation that is shown in black. The standard form for the first order polynomial is

$$\begin{aligned} y &= P(1) \cdot x + P(2) \\ where: P(1) &= -0.831726 \\ P(2) &= -0.0204933 \end{aligned} \tag{1}$$

and when the log of x and y are taken, it becomes

$$\begin{aligned} \log_{10}(y) &= P(1) \cdot \log_{10}(x) + P(2) \\ y &= 10^{(P(1)\cdot\log_{10}(x)+P(2))}. \end{aligned} \tag{2}$$

Solving Equation 2 for x provides a simple expression to calculate the number of nodes required to reduce an image render time to the fraction of time y that it takes to render the image on a single node,

$$x = 10^{(\frac{\log_{10}(y)-P(2)}{P(1)})}. \tag{3}$$

Equation 3 is dependant on the fact that all nodes in the cluster are equal in performance, and that no other external influences, such as network congestion, are present. This equation would also likely vary from application to application, based on the processes I/O requirements.

Adding nodes to a cluster will not always help to improve the parallel system performance. Eventually, enough systems will be sharing the network, such that the bandwidth will become a limiting factor in the effectiveness of an OpenMosix cluster. We tested our system by analyzing the network traffic during system load while running ppovray on 2, 5, and 9 nodes in the cluster. With all 9 nodes executing POV-Ray in parallel, the maximum network utilization is only 1.2 Mbps. This value is certainly well within our maximum bandwidth. The traffic analysis only becomes interesting when you examine the network for the entire duration of the process (i.e. from process creation to migration to termination). Figure 6 shows the network traffic over 60 seconds while nine nodes render our test image. The processes begins at 0 seconds and begins to migrate at 3 seconds. The network quickly reaches the maximum capacity of MFS as processes are packed up and shipped out to other nodes. The network traffic then becomes relatively idle until the processes finish and are sent back to the originating node.

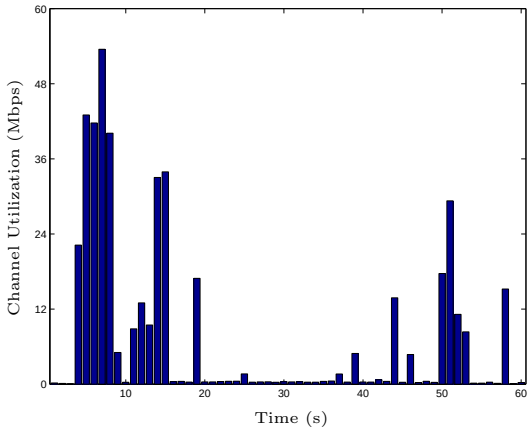

Figure 6: Network Traffic for a Full POV-Ray Render

So, from Figure 6 it is obvious that we are fully utilizing the network bandwidth, at least when dealing with MFS. However, this only occurs at the beginning, when many processes are being sent to other nodes all at once. As the processes run there is very little network load. Increasing the number of nodes, and likewise the number of simultaneous processes would most likely increase the duration of this heavy network load startup zone. If the running processes then finish executing within seconds, then there is a large waste of bandwidth and time. OpenMosix and parallel processing in general is intended for applications that run for a substantial amount of time when compared with the time it takes for processes to migrate. OpenMosix will be more efficient when applications take minutes or even hours to process, because at this point, the setup time is negligible.

5.3 Testing Sample Application: LAME MP3 Encoder

As stated before, another limiting factor on the number of nodes is the level of I/O required by a migrated application. Our final test, to cover this scenario, consisted of ripping wave files to MP3s using the LAME MP3 encoder [10]. In this test we used all nine nodes to rip nine identical wave files to MP3s in parallel. In addition to the heavy load during process migration, there is nearly a 20 Mbps constant bandwidth usage for the duration of the encoding procedure as shown in Figure 7. At the rate shown, a cluster roughly five times the size of ours would completely saturate the network and transform this CPU bound application into an I/O bound process. Network intensive applications such as this become the limiting factor in determining how many nodes the cluster is able to efficiently support. However, it is possible to extend this barrier by upgrading the network infrastructure to options including Gigabit Ethernet and Myrinet.

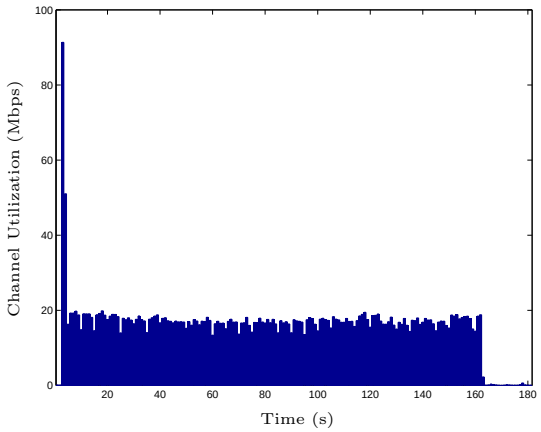

Figure 7: Network Traffic During MP3 Encoding

6 Conclusions

Overall this project was a very informative lesson in cluster computing. We successfully linked ten machines to form a large scale parallel computing environment. The real advantages in using a package such as OpenMosix is that processes migrate transparently and that it is as simple as applying a Linux kernel patch. Other clustering environments require special software to be written to take advantage of networked parallelism. However, with OpenMosix existing applications can be made to migrate across multiple machines by simply forking within the program (creating a duplicate process from the one currently running) or running them multiple times. The end user can automate this process by writing simple scripts that take advantage of easily parallelizable applications that are run multiple times and especially those that run several instances at once.

From the tests performed, it is apparent that running multiple OpenMosix nodes greatly enhances computing capabilities. The degree to which the computing is improved depends upon the target application. Certain disk access intensive applications cause a diskless cluster to reach an expansion barrier that would not be reached with a non-disk access intensive application.

We are intending to continue work on this project, building a ClusterFLOP II from components that are more representative of what corporations [not limited to a graduate student budget] might use to implement such a computing cluster. We would see a huge performance increase simply from using processors with FPUs that run at the full speed of the chip, as opposed to the 1/4 speed FPUs that were implemented on our VIA C3s.

References

[1] VIA Technologies, Inc. "VIA C3^{TM} in EBGA; Datasheet" July 25, 2003, viewed April 2004 <http://www.via.com.tw/en/Products/C3/C3_EGBA_datasheet.zip>

[2] Gutschke, Markus, et al. "Welcome to EtherBoot.org" viewed April 2004 <http://etherboot.com/>

[3] ROM-o-matic.net. "ROM-o-matic.net" viewed April 2004 <http://rom-o-matic.net/>

[4] American Megatrends Inc. "AMI: American Megatrends Inc.: Home Page" viewed April 2004 <http://www.ami.com/>

[5] CrazyApe. "AMIBCP Repository" viewed April 2004 <http://www.stormpages.com/crazyape/amibcp.html/>

[6] Amersdorfer, Markus. "HOWTO set up a Network with Diskless Workstations using Debian GNU/Linux" viewed April 2004 <http://homex.subnet.at/ max/diskless/index.php>

[7] Combs, Gerald, et al. "Ethereal: A Network Protocol Analyzer" viewed April 2004 <http://www.ethereal.com/>

[8] Buytaert, Kris, et al. "The openMosix HOWTO" viewed April 2004 <http://howto.ipng.be/openMosix-HOWTO/>

[9] POV-Team. "POV-Ray - The Persistence of Vision Raytracer" viewed April 2004 <http://www.povray.org/>

[10] Cheng, Mike, et al. "LAME Ain't an MP3 Encoder" viewed April 2004 <http://lame.sourceforge.net/>

[11] Michlmayr, Martin, et al. "Debian GNU/Linux – The Universal Operating System" viewed April 2004 <http://www.debian.org/>

[12] Bar, Moshe, et al. "openMosix, an Open Source Linux Cluster Project" viewed April 2004 <http://openmosix.sourceforge.net/>

[13] Syntax Groups Corporation. "Syntax USA" viewed April 2004 <http://www.syntaxusa.com/>

Powered by LaTeX