A Linux PC Cluster with Diskless Slave Nodes for Parallel Computing

Chao-Tung Yang and Yao-Chung Chang

High-Performance Computing Laboratory
Department of Computer Science and Information Engineering
Tunghai University
Taichung, 407, Taiwan, R.O.C.
Tel: +886-4-23590121 ext. 3279
Fax: +886-4-23591567
ctyang@mail.thu.edu.tw

Abstract. *This document describes how to set up a diskless Linux box. As technology is advancing rapidly, network-cards are becoming cheaper and much faster - 100 Mbits/s Ethernet is common now and in about 1 year 1000 Mbits/s i.e. 1GigBits Ethernet cards will become a standard. With high-speed network cards, remote access will become as fast as the local disk access which will make diskless nodes a viable alternative to workstations in local LAN. Also diskless nodes eliminate the cost of software upgrades and system administration costs like backup, recovery which will be centralized on the server side. Diskless nodes also enable "sharing/optimization" of centralized server CPU, memory, hard-disk, tape and CDROM resources. Diskless nodes provides mobility for the users i.e., users can log on from any one of diskless nodes and are not tied to one workstation. In this paper, a SMP-based PC cluster consists of one master node and eight diskless slave nodes (16 processors), is proposed and built. The system architecture and benchmark performances of the cluster are also presented in this paper.*

1 Introduction

Extraordinary technological improvements over the past few years in areas such as microprocessors, memory, buses, networks, and software have made it possible to assemble groups of inexpensive personal computers and/or workstations into a cost effective system that functions in concert and posses tremendous processing power. Cluster computing is not new, but in company with other technical capabilities, particularly in the area of networking, this class of machines is becoming a high-performance platform for parallel and distributed applications [1, 2, 8, 9].

In traditional scalable computing clusters, there is related high powered Intel or AMD based PC's associated with several gigabytes of hard disk space spread around. As far as a cluster administrator is concerned, the installer may have to install an operating system and related software for each cluster node. Every node (excluding the Server) is equipped without any hard disk and is booted from the network. Since it has booted successfully, it works as same as a fully equipped one. Although, users can store their own data on the local hard drives in this way, backups and other software installation are rarely (even never) performed for computing purpose. As a result, we can see drawbacks that much storage space is wasted and we have to take care of each node in front of the console if something is going wrong. Does it really make sense to have a full computer for each node? It's not necessary. With the development of NOW (Networks of workstations), it has been applied to computing clusters also. Every node (excluding the Server) is equipped without any hard disk and is booted from the network. Since it has booted successfully, it works as same as a fully equipped one. We will explain how to set up a diskless cluster for computing purpose.

Motivations for the use of diskless Linux within the Department of CSIE are listed as bellows:

- To manage a large number of workstations in a consistent and efficient manner. Management includes software updates bug fixes, system configuration, file system integrity, security etc.
- To control a group of computers form a central location. These computers could be embedded processors or large numbers of computers ganged together to solve compute intensive problems.
- Lab computers are subject to a wide range of abuses that cause the system disk to become corrupted. This could be as a result of powering off or resetting the computer without properly shutting down the computer.
- Upgrading system software without worrying about whether the machine is turned on or not.
- Improved system security through the use of read only file systems and verification programs such as tripwire that ensure the integrity of the image that is presented to the user.

The use of loosely coupled, powerful and low-cost commodity components (PCs or workstations, typically), especially without any hard disk drive, connected by high-speed networks has resulted in the

widespread usage of a technology popularly called diskless cluster. Strictly speaking, it consists of one or more servers which provide not only bootstrap service but also related network services (such as DHCP, NIS, NFS servers, and etc) and many clients with no hard disk drive requesting for booting from the network. The availability of such clusters made maintain as easy as possible, and also reduced the waste in storage space. The diskless cluster differs from the traditional one in that a network is used to provide not only inter-processor communications but also a medium for booting and transmission for a live file system. Thus, each diskless node before booting can boot through a floppy disk or a NIC's boot ROM with a small bootstrap program and even with a NIC's PXE, which sends a broadcast packet to a DHCP server and is then assigned an IP address. After each node has been assigned a valid IP address, it sends a request to the TFTP server for getting the boot image, referred to the Linux Kernel, through TCP/IP protocol and starts the booting process. During the booting process, all the necessary system files are transmitted through the network. After the remote file system is mounted as root file system (NFS_ROOT), and the system initialization is done, the node is ready to work.

This document describes how to set up a diskless Linux box. As technology is advancing rapidly, network-cards are becoming cheaper and much faster - 100 Mbits/s Ethernet is common now and in about 1 year 1000 Mbits/s i.e. 1GigBits Ethernet cards will become a standard. With high-speed network cards, remote access will become as fast as the local disk access which will make diskless nodes a viable alternative to workstations in local LAN. Also diskless nodes eliminate the cost of software upgrades and system administration costs like backup, recovery which will be centralized on the server side. Diskless nodes also enable "sharing/optimization" of centralized server CPU, memory, hard-disk, tape and CDROM resources. Diskless nodes provides mobility for the users i.e., users can log on from any one of diskless nodes and are not tied to one workstation. Diskless Linux box completely eliminates the need for local floppy disk, CDROM drive, tape drive and hard-disk. Diskless nodes JUST has a network card, RAM, a low-end CPU and a very simple mother-board which does not have any interface sockets/slots for hard disks, modem, CDROM, floppy etc.. With Diskless Linux nodes you can run programs on remote Linux 64 CPU SMP box or even on Linux super-computer! Diskless nodes lower the "Total Cost of Ownership" of the computer system.

In this paper, a SMP-based PC cluster consists of one master node and eight diskless slave nodes (16 processors), is proposed and built. In this paper, the system architecture and benchmark performances of the cluster are presented. In order to measure the performance of our cluster, the parallel ray-tracing problem is illustrated and the experimental result is demonstrated on our Linux SMPs cluster. The experimental results show that the highest speedup is 15.22 for PVMPOV [5, 7], when the total numbers of processor is 16 on SMPs cluster. Also, the LU of NPB benchmark is used to demonstrate the performance of our cluster tested by using LAM/MPI library [4]. The experimental result shows that our cluster can obtain speedup 9.07 when the total numbers of processors used is 16. The results of this study will make theoretical and technical contributions to the design of a high-performance computing system on a Linux SMP Clusters with diskless slave nodes.

2 System Setup

2.1 Description of Hardware and System Software

Our SMP cluster is a low cost Beowulf-type class supercomputer that utilizes multi-computer architecture for parallel computations. The cluster as shown in consists of nine PC-based symmetric multiprocessors (SMP) connected by one 24-port 100Mbps Ethernet switch with Fast Ethernet interface. Its system architecture is shown in. There are one server node and eight computing nodes. The server node has two Intel Pentium-III 690MHz (550 over-clock, FSB 138MHz) processors and 256MBytes of shared local memory. Each Pentium-III has 32K on-chip instruction and data caches (L1 cache), a 256K on-chip four-way second-level cache with full speed of CPU. The other eight nodes are Celeron-based SMP machines. Each Celeron also has 32K on-chip instruction and data caches (L1 cache), a 128K on-chip four-way second-level cache with full speed of CPU. Each individual processor is rated at 495MHz, and the system bus has a clock rate of 110 MHz.

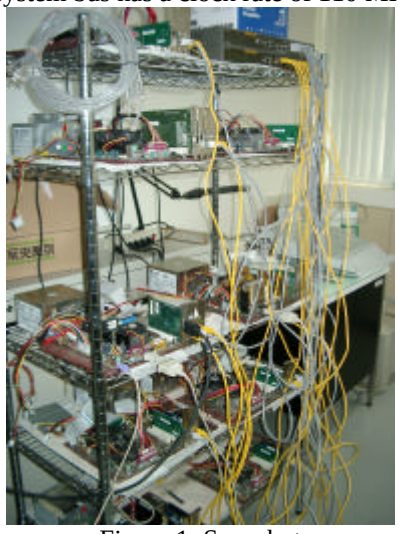

Figure 1: Snapshot

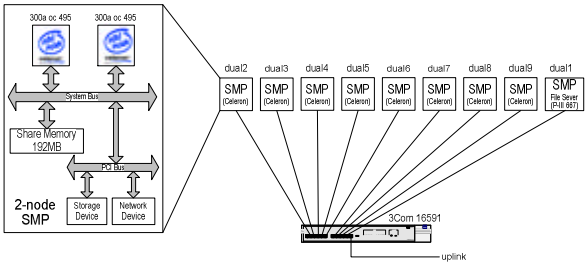

Figure 2: System overview

Linux is a robust, free and reliable POSIX compliant operating system. Several companies have built businesses from packaging Linux software into organized distributions; RedHat is an example of such a company. Linux provides the features typically found in standard UNIX such as multi-user access, pre-emptive, multi-tasking, demand-paged virtual memory and SMP support. In addition to the Linux kernel, a large amount of application and system software and tools are also freely available. This makes Linux the preferred operating system for clusters. The idea of the Linux cluster is to maximize the performance-to-cost ratio of computing by using low-cost commodity components and free-source Linux and GNU software to assemble a parallel and distributed computing system. Software support includes the standard Linux/GNU environment, containing compilers, debuggers, editors, and standard numerical libraries. Coordination and communication among the processing nodes is a key requirement of parallel-processing clusters. In order to accommodate this coordination, developers have created software to carry out the coordination and hardware to send and receive the coordinating messages. Messaging architectures such as MPI or Message Passing Interface, and PVM or Parallel Virtual Machine, allow the programmers to ensure that control and data messages take place as needed during operation.

PVM, or Parallel Virtual Machine, started out as a project at the Oak Ridge National Laboratory and was developed further at the University of Tennessee. PVM is a complete distributed computing system, allowing programs to span several machines across a network. PVM utilizes a Message Passing model that allows developers to distribute programs across a variety of machine architectures and across several data formats. PVM essentially collects the network's workstations into a single virtual machine. PVM also allows a network of heterogeneous computers to be used as a single computational resource called the parallel virtual machine. As we have seen, PVM is a very flexible parallel processing environment. It therefore supports almost all models of parallel programming, including the commonly used all-peers and master-slave paradigms.

MPI is a message-passing application programming interface with protocol and semantic specifications for how its features must behave in any implementation (such as a message buffering and message delivery progress requirement). MPI includes point-to-point message passing and collective (global) operations. These are all scoped to a user-specified group of processes. MPI provides a substantial set of libraries for the writing, debugging, and performance testing of distributed programs. Our system currently uses LAM/MPI, a portable implementation of the MPI standard developed cooperatively by Notre Dame University. LAM (Local Area Multicomputer) is an MPI programming environment and development system and includes a visualization tool that allows a user to examine the state of the machine allocated to their job as well as provides a means of studying message flows between nodes.

2.2 Set up Hardware

Here are the main steps in setting up the hardware:

1. Move the machines from various labs in the campus to the high-performance computing Lab and mount them on the rack, name and number them as well.
2. Setup the network switch and connect each port to Fast Ethernet adapters of the machines one by one.
3. A single monitor, keyboard and mouse is connected to one of the machines, referred to server node (dual1), that are responsible for bootstrap service and related network services with a hard disk drive (30GB) and a high-end graphics card.
4. The other diskless machines, referred to client nodes (dual2~dual9), are equipped with a 1.44MB floppy disk drive and no graphics card.
5. Power up the server node (dual1).

2.3 Set up Software

These instructions outline the procedure required to set-up Linux diskless boot using RedHat 7.2, dhcpd & etherboot on a system using a RealTek RTL8139 Fast Ethernet Network Interface Cards (NICs). The diskless workstation will be able to boot from a server node and mount remote file systems. Adapting to other NIC's will not be very difficult to do, and it's almost the same.

1. OS installation: RedHat Linux 7.2 is installed on the server node by connecting all the peripherals such as monitor, mouse and keyboard. Most of the hardware was automatically detected, so the main focus is on partitioning the drive and choosing the relevant packages to be installed. It is very important to choose partition size which is correct for the need because it might be very difficult to change this at a later stage when the cluster will be in the functional mode. Following is the list of partitions:

- The `/` partition is about 5GB. This / partition contains `/bin`, `/boot`, `/dev`, `/lib`, `/root`, `/sbin`, `/var` and especially the `/tftproot` which contains boot images and copies of a live file system for each client node.
- The `/usr` partition is about 15GB. This `/usr` partition is created by keeping in mind that most additional rpm packages will be installed in `/usr`.

- The swap partition: Swapping is really bad for the performance of the system. Unfortunately there might be chance when the machine is computing a very large job and just don't have enough memory. Since the machines have 256MB RAM, it is realized that a 256MB of swap partition is a good idea. On the other hand, we don't enable swap on client nodes, because swap over NFS is a bottleneck of network performance.
- The `/home` partition: Rest part of the disk. This partition is used as a common home area for users on individual client nodes through NFS.
- Network Configuration: During OS installation IP addresses and node names are assigned. Following are the server node name of Diskless Beowulf Cluster dual1 (192.168.1.1).
- For the configuration, the following files are modified: `/etc/sysconfig/network` and `/etc/sysconfig/network-scripts/ifcfg-eth0`. Here are the contents of these two files

```
/etc/sysconfig/network
NETWORKING=yes
HOSTNAME=dual1
NISDOMAIN=dual

/etc/sysconfig/network-
scripts/ifcfg-eth0
DEVICE=eth0
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.1.1
NETMASK=255.255.255.0
```

- `/etc/hosts.equiv`: In order to allow remote shells (**`rsh`**) from any node to any other in the cluster, for all users, we should relax the security, and list all hostnames in `/etc/hosts.equiv`. Here we insert dual1 ~ dual8 into individual lines

2. Run **`ntsysv`** command to enable minimum services, because we will make a copy of a live file system as same as server node for diskless client nodes in order to avoid unnecessary services for client nodes to start in boot time. The follows are the services that we recommanded:

```
[*]anacron
[*]atd
[*]crond
[*]keytable
[*]network
[*]portmap
[*]rlogin
[*]rsh
[*]sshd
[*]syslog
[*]xinetd
[*]netfs
```

3. Make several copies of a live file system of server node into `/tftpboot/` directory. First of all, we create `/tftpboot/dual2`, `/tftpboot/dual3`, …, and `/tftpboot/dual9` directories, respectively, then perform the following steps for individual directory:

```
cd /
umask 022
mkdir -p /tftpboot/dual2/bin
mkdir -p /tftpboot/dual2/mnt
mkdir -p /tftpboot/dual2/proc
mkdir -p /tftpboot/dual2/tmp
mkdir -p /tftpboot/dual2/usr
mkdir -p /tftpboot/dual2/home

chmod 1777 /tftpboot/dual2/tmp
cp -a bin lib sbin dev etc root
var /tftpboot/dual2
```

Repeat the similar steps for `/tfptboot/dual3`, `/tftpboot/dual4`, etc. These commands make copies of the file system files that will be used by the client in minimum. Before proceeding, be sure you are working on the COPIES not the originals. It is very easy to make such mistake if not careful

4. Modify the following configuration file for each node in `/tftpboot/dual[2-9]/etc/` `/etc/sysconfig/network` and /`etc/sysconfig/network-scripts/ifcfg-eth0`, especially the file /`etc/fstab`

```
dual1:/tftpboot/dual2/      nfs
    defaults          0 0
dual1:/usr/usr        nfs
    defaults          0 0
dual1:/home     /home       nfs
    defaults          0 0
none      /dev/pts    devpts
    gid=5,mode=620    0 0
none      /proc       proc
    defaults          0 0
none      /dev/shm    tmpfs
    defaults          0 0
```

At firth line, `dual1:/tftpboot/daul2/` may be `dual1:/tftpboot/daul3`, …, `dual1:/tftpboot/dual9` regard to which directory you are in.

5. Install TFTP daemon rpm package: After we have installed TFTP daemon rpm packages, be sure the associated file `/etc/xinetd.d/tftp` is as follows

```
service tftp
{
disable         = no
socket_type     = dgram
protocol        = udp
wait      = yes
```

```
user      = root
server          =
/usr/sbin/in.tftpd
server_args     = -s /tftpboot
}
```

6. Install DHCP daemon rpm package (version 3), and modify associated configuration file as follows. Note: For DHCP daemon to start correctly, you should enable the kernel option (CONFIG_NETLINK_DEV).

```
# Configuration the file for ISC
dhcpd v3.0
not authoritative;
ddns-update-style     none;       #
required for ISC v3.0

shared-network DISKLESS-BEOWULF {
      subnet  192.168.1.0  netmask
255.255.255.0 {
      }
}
group     {

      option domain-name
    "hpc.csie.thu.edu.tw";
      option subnet-mask
    255.255.255.0;
      option      broadcast-address
    192.168.1.255;
      use-host-decl-names
    on;


      host dual1 {
            hardware       ethernet
    00:90:cc:0b:79:54;
           fixed-address
    192.168.1.1;
           if substring (option
vendor-class-identifier, 0, 9) =
"PXEClient" {
            filename
          "/vmlinuz.pxe";
           }else if substring
(option  vendor-class-identifier,  0,
9) = "Etherboot" {
             filename
"/vmlinuz.nbi";
           }

      }
      host dual2 {
            hardware      ethernet
    00:90:cc:0b:79:55
           fixed-address
    192.168.1.2;
           if substring (option
vendor-class-identifier, 0, 9) =
"PXEClient" {
            filename
          "/vmlinuz.pxe";
           }else if substring
(option  vendor-class-identifier,  0,
9) = "Etherboot" {
             filename
"/vmlinuz.nbi";
           }
      }
} # group
```

The job of DHCP daemon is to provide boot from Ethernet for diskless client nodes. Where hardware Ethernet is one of the MAC addresses of NICs among diskless client nodes .It is unique to each NIC and important to the process of diskless boot, because each node at boot time is assigned a specific IP address by judging from the MAC address, then preceding the subsequent process.

7. Network File System configuration: We have used fully local OS install configuration for the server node of diskless Beowulf cluster. So far, all the client nodes are sill useless lumps of metal without operating systems, all we have to do is set up NFS for providing remote boot and root file system mount. Following were the steps used to configure NFS:

- We selected dual1 as server node and exported the `/home`, `/usr`, and `/tftproot/dual*` directory which contains the live file systems and boot images by modifying the `/etc/exports` of server node as :

```
/tftpboot/dual2
    dual2(rw,no_root_squash)
/tftpboot/dual3
    dual3(rw,no_root_squash)
/tftpboot/dual4
    dual4(rw,no_root_squash)
/tftpboot/dual5
    dual5(rw,no_root_squash)
/tftpboot/dual6
    dual6(rw,no_root_squash)
/tftpboot/dual7
    dual7(rw,no_root_squash)
/tftpboot/dual8
    dual8(rw,no_root_squash)
/tftpboot/dual9
    dual9(rw,no_root_squash)
/usr
192.168.1.1/255.255.255.0(ro,no_ro
ot_squash)
/home
      192.168.1.1/255.255.255.0(rw
      ,no_root_squash)
```

Make sure your NFS daemon of server node is enabled at boot time

8. Network Information System configuration: The Network Information Service (NIS) is an

administrative database that provides central control and automatic dissemination of important administrative files. NIS converts several standard UNIX files into databases that can be queried over the network. The databases are called NIS maps. Following are the main steps for configuring NIS:

- Configuration of NIS master server: We selected NIS domain name "diskless" by issuing the command **authconfig** and the options that we choose are as follows

```
[*] Use NIS
Domain: dual
```

- Then we initiated the yp services as

```
#/etc/init.d/ypserv start
#cd /var/yp
#make
```

Make sure to rebuild the NIS database after creating user accounts by issuing **make** in /var/yp directory

9. Make a Linux bootable kernel for each client node: When make a new kernel with make **make config**, please be sure to enable the following options

```
IP: kernel level autoconfiguration
(CONFIG_IP_PNP) [N/y/?] y
IP: DHCP support
(CONFIG_IP_PNP_DHCP) [N/y/?] (NEW)
y
IP: BOOTP support
(CONFIG_IP_PNP_BOOTP) [N/y/?]
(NEW) y
IP: RARP support
(CONFIG_IP_PNP_RARP) [N/y/?] (NEW)
y
NFS file system support
(CONFIG_NFS_FS) [Y/m/n/?] y
Provide NFSv3 client support
(CONFIG_NFS_V3) [Y/n/?] y
(The server node should enable
above options also)
Root file system on NFS
(CONFIG_ROOT_NFS) [Y/n/?] y
(The server node should not enable
this option)
```

- The last line above is quite important, because the diskless client nodes will mount remote file system as root file system through NFS export points. Also, we should make the kernel of clients as simple as possible. That's to say, no modules option is required.
- After compilation, we have to tag a kernel (usually named bzImage) for etherboot. First of all, install the tool **mknbi** through RPM package, and type the following command as root user

```
# mknbi-linux --ipaddrs=rom \
/usr/src/linux/arch/i386/boot/bzIm
age > /tftpboot/vmlinuz.nbi
```

10. Make a ether bootstrap floppy disk or PXE boot image for each diskless client nodes:
    - Marty Connor has set up form (http://rom-o-matic.net/) for creating a ROM image on the fly and returning it as the output file of image. If all you want is just a ROM image, this could save you time building the floppy disk. When you get an output image such as eb-5.0.4-rtl8139.lzdsk, type the following command for each floppy disk

```
dd      if=./eb-5.0.4-rtl8139.lzdsk
of=/dev/fd0
```

    - And insert the floppy disks to each diskless client nodes. If you want to use PXE boot (your NICs must support) without using any floppy disk for each client, please create an output image with PXE support (suffix with .lzpxe). And copy the image as `/tftpboot/vmlinuz.pxe`, then enable the NIC's PXE boot of each client.

11. BIOS configuration: For booting machines without monitor, keyword and mouse, BIOS was configured on all the machines. We connect the monitor, mouse and keyboard to the nodes and configure the BIOS for no halt in the absence of keyboard, mouse and monitor.

12. Message passing libraries installation: Installation is automatic. During the selection of packages, By selecting clustering tools and it will install in `/usr/bin/mpicc` and `/usr/bin/mpif77`. PVM installation: Parallel Virtual Machine (PVM) installation is automatic as well. During the selection of packages, by selecting clustering tools and it will install in `/usr/share/pvm3`.

13. Run **ntsysv** command for the server node to enable NFS server, NIS server, DHCP daemon and TFTP service, and then reboot.

14. After the server node has booted, power up all the client nodes. You may see the following similar boot message through a serial console (you must enable it at step 9 and 10) connected to a dumb terminal.

```
Searching for server (DHCP)...
Me:      192.168.1.2,      Server:
192.168.1.1, Gateway:
Loading
192.168.1.1:/tftpboot/vmlinuz.nbi
(NBI)... done
mknbi-1.2-2/first32.c (GPL)
129792k total memory
Uncompressing Linux... Ok, booting
the kernel
```

```
Linux version 2.4.18 (root@dual1)
gcc version 2.96 20000731 (Red Hat
Linux 7.3 2.96-110)) #1 Thu Apr 5
15:18:02 EST 2002
BIOS-provided physical RAM map:
BIOS-e820:   00090000   @   00000000
(usable)
BIOS-e820:   07dc0000   @   00100000
(usable)
Detected 494.961 Mhz processor.
```

Finally, a Login prompt will occur, then telnet into each client nodes and run **`authconfig`** to enable NIS and set up its NIS domain and NIS server. If you want to start up additional services at boot time, you may run **`ntsysv`** command to enable what you want.

15. Test if everything is going to work fine, and enjoy your high performance computing

3 Performance Results

3.1 Matrix Multiplications

The biggest price we had to pay for the use of a PC cluster was the conversion of an existing serial code to a parallel code based on the message-passing philosophy. The main difficulty with the message-passing philosophy is that one needs to ensure that a control node (or master node) is distributing the workload evenly between all the other nodes (the compute nodes). Because all the nodes have to synchronize at each time step, each PC should finish its calculations in about the same amount of time. If the load is uneven (or if the load balancing is poor), the PCs are going to synchronize on the slowest node, leading to a worst-case scenario. Another obstacle is the possibility of communication patterns that can deadlock. A typical example is if PC A is waiting to receive information from PC B, while B is also waiting to receive information from A.

The matrix operation derives a resultant matrix by multiplying two input matrices, **a** and **b**, where matrix **a** is a matrix of *N* rows by *P* columns and matrix **b** is of *P* rows by *M* columns. The resultant matrix **c** is of *N* rows by *M* columns. The serial realization of this operation is quite straightforward as listed in the following:

```
for(k=0; k<M; k++)
    for(i=0; i<N; i++){
          c[i][k]=0.0;
          for(j=0; j<P; j++)

    c[i][k]+=a[i][j]*b[j][k];
    }
```

Its algorithm requires n^3 multiplications and n^3 additions, leading to a sequential time complexity of $\boldsymbol{O}(n^3)$. Let's consider what we need to change in order to use PVM. The first activity is to partition the problem so each slave node can perform on its own assignment in parallel. For matrix multiplication, the smallest sensible unit of work is the computation of one element in the result matrix. It is possible to divide the work into even smaller chunks, but any finer division would not be beneficial because of the number of processor is not enough to process, i.e., n^2 processors are needed.

The matrix multiplication algorithm is implemented in PVM using the master-slave paradigm. The master task is named `master_mm_pvm`, and the slave task is named `slave_mm_pvm`. The master reads in the input data, which includes the number of slaves to be spawned, *nTasks*. After registering with PVM and receiving a *taskid* or *tid*, it spawns *nTasks* instances of the slave program `slave_mm_pvm` and then distributes the input graph information to each of them. As a result of the spawn function, the master obtains the *tids* from each of the slaves. Since each slave needs to work on a distinct subset of the set of matrix elements, they need to be assigned instance IDs in the range (0... *nTask-1*). The *tids* assigned to them by the PVM library do not lie in this range, so the master needs to assign the instance IDs to the slave nodes and send that information along with the input matrix. Each slave also need to know the total number of slaves in the program, and this information is passed on to them by the master process as an argument to the spawn function since, unlike the instance IDs, this number is the same for all *nTasks* slaves.

The matrix multiplication was run with forking of different numbers of tasks to demonstrate the speedup. The problem sizes were 256×256, 512×512, 1024×1024, and 2048×2048 in our experiments. It is well known, the speedup can be defined as *Ts/Tp*, where *Ts* is the execution time using serial program, and *Tp* is the execution time using multiprocessor. The execution times and corresponding speedups by using sixteen processors with different problem sizes were listed in **Figure 3** and **Figure 4**, respectively. In, the corresponding speedup is increased for different problem sizes by varying the number of slave programs.

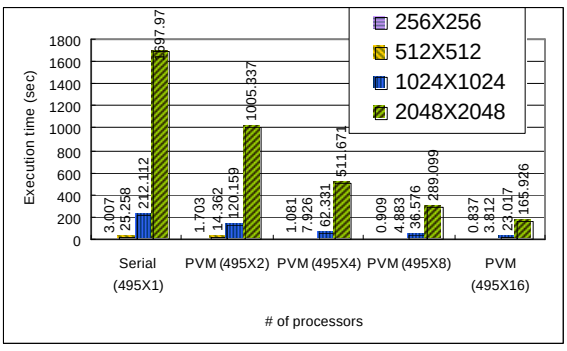

Figure 3: The execution time of MM using processor form 1 to 16.

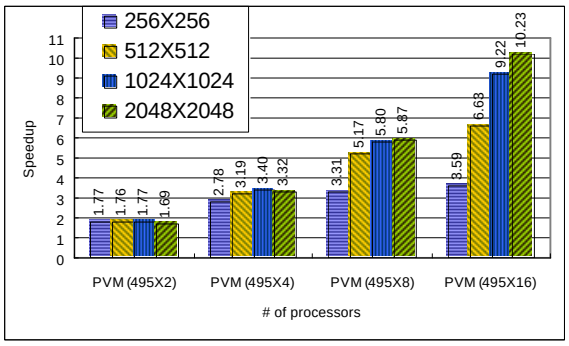

Figure 4: The speedup of MM

3.2 NAS Parallel Benchmark

The NAS Parallel Benchmark (NPB) is a set of 8 programs designed to help evaluate the performance of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications. NPB 2.3 is MPI-based source-code implementations written and distributed by NAS. They are intended to run with little or no tuning, and approximate the performance a typical user can expect to obtain for a portable parallel program. The LU benchmark is based on the NX reference implementation from 1991. This code requires a power-of-two number of processors. A 2-D partitioning of the grid onto processors occurs by halving the grid repeatedly in the first two dimensions, alternately x and then y, until all power-of-two processors are assigned, resulting in vertical pencil-like grid partitions on the individual processors. This ordering of point based operations constituting the SSOR procedure proceeds on diagonals which progressively sweep from one corner on a given z plane to the opposite corner of the same z plane, thereupon proceeding to the next z plane. Communication of partition boundary data occurs after completion of computation on all diagonals that contact an adjacent partition. This constitutes a diagonal pipelining method and is called a "wavefront" method. It results in relatedly large number of small communications of 5 words each.

A NAS benchmark that we chose to present here is LU. For the LU benchmark, the sizes were class A and B. The execution time of LU was shown in Figure 5. The performance numbers for 16 processors as reported in Figure 5 by the LU benchmark were 715.06 MFLOPS and 778.62 MFLOPS for class A and class B, respectively. As a measure of scalability, we selected parallel speedup, as classically calculated. The serial time was obtained by running the benchmarks on one processor. The speedup of LU benchmark is reported in Figure 5.

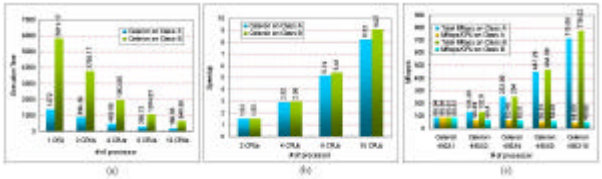

Figure 5: (a) Execution time of LU. (b) Speedup of LU using 16 processors. (c) Total MFlops/s obtained using 16 processors.

3.3 PVMPOV for Parallel Rendering

Rendering is a technique for generating a graphical image from a mathematical model of a two or three-dimensional object or scene. A common method of rendering is ray tracing. Ray tracing is a technique used in computer graphics to create realistic images by calculating the paths taken by rays of light entering the observer's eye at different angles. Ray tracing is an ideal application for parallel processing since there are many pixels, each of whose values are independent and can be calculated in parallel. The Persistence of Vision Ray Tracer (POV-Ray) is an all-round 3-dimensional ray tracing software package [5]. It takes input information and simulates the way light interacts with the objects defined to create 3D pictures and animations. In addition to the ray tracing process, newer versions of POV can also use a variant of the process known as radiosity (sophisticated lighting) to add greater realism to scenes, particularly those that use diffuse light POVRay can simulate many atmospheric and volumetric effects (such as smoke and haze).

Given a number of computers and a demanding POVRay scene to render, there are a number of techniques to distribute the rendering among the available resources. If one is rendering an animation then obviously each computer can render a subset of the total number of frames. The frames can be sent to each computer in contiguous chunks or in an interleaved order, in either case a preview (every Nth frame) of the animation can generally be viewed as the frames are being computed. POVRay is a multi-platform, freeware ray tracer. Many people have modified its source code to produce special "unofficial" versions. One of these unofficial versions is PVMPOV, which enables POVRay to run on a Linux cluster.

PVMPOV has the ability to distribute a rendering across multiple heterogeneous systems. Parallel execution is only active if the user gives the "+N" option to PVMPOV. Otherwise, PVMPOV behaves the same as regular POV-Ray and runs a single task only on the local machine. Using the PVM code, there is one master and many slave tasks. The master has the responsibility of dividing the image up into small blocks, which are assigned to the slaves. When the slaves have finished rendering the blocks, they are sent back to the master, which combines them to form the final image. The master does not render anything by itself, although there is usually a slave running on

the same machine as the master, since the master doesn't use very much CPU power.

If one or more slaves fail, it is usually possible for PVMPOV to complete the rendering. PVMPOV starts the slaves at a reduced priority by default, to avoid annoying the users on the other machines. The slave tasks will also automatically time out if the master fails, to avoid having lots of lingering slave tasks if you kill the master. PVMPOV can also work on a single machine, like the regular POV-Ray, if so desired. The code is designed to keep the available slaves busy, regardless of system loading and network bandwidth. We have run PVMPOV on our 16-Celeron and 16-PIII processors testbed and have had amazing results, respectively. With the cluster configured, runs the following commands to begin the ray tracing and generates the image files as shown in Figure 6.

```
$pvmpov +iskyvase.pov +w640 +h480 +nt16 -L/home/ct/pvmpov3_1g_2/povray31/include
$pvmpov +ifish13.pov +w640 +h480 +nt16 -L/home/ct/pvmpov3_1g_2/povray31/include
$pvmpov +ipawns.pov +w640 +h480 +nt16 -L/home/ct/pvmpov3_1g_2/povray31/include
$pvmpov +iEstudio.pov +w640 +h480 +nt16 -L/home/ct/pvmpov3_1g_2/povray31/include
```

This is the benchmark option command-line with the exception of the +nw and +nh switches, which are specific to PVMPOV and define the size of image each of the slaves, will be working on. The +nt switch is specific to the number of tasks will be running. For example, +nt16 will start 16 tasks, one for each processor. The messages on the screen should show that slaves were successfully started. When completed, PVMPOV will display the slave statistics as well as the total render time. In case of Skyvase model, by using single Celeron processor mode of a dual processor machine for processing 1600X1280 image, the render time was 256 seconds. Using out Celeron-based SMP cluster (16 processors) further reduced the time to 26 seconds. The execution times for the different POVray model (Skyvase, Fish13, Pawns, and Estudio) on Celeron SMPs and P-III SMP clusters were shown in Figure 7, respectively. The corresponding speedups of different problem size by varying the number of task (option: +nt) was shown in Figure 8. The highest speedups were obtained about 15.22 (1600X1280) for Pawns model by using our Celeron SMPs cluster with 16 processors.

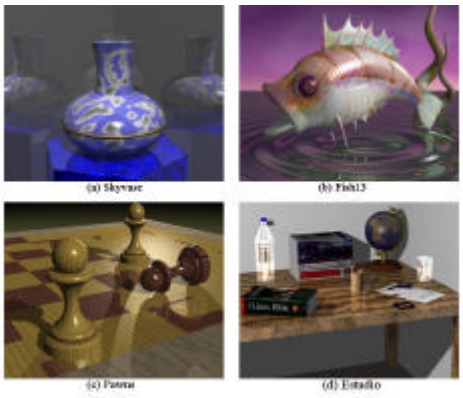

Figure 6: Four diagrams were generated by PVMPOV.

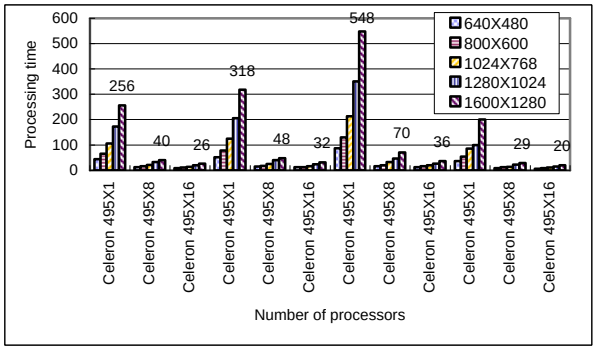

Figure 7: Execution times of PVMPOV diagram.

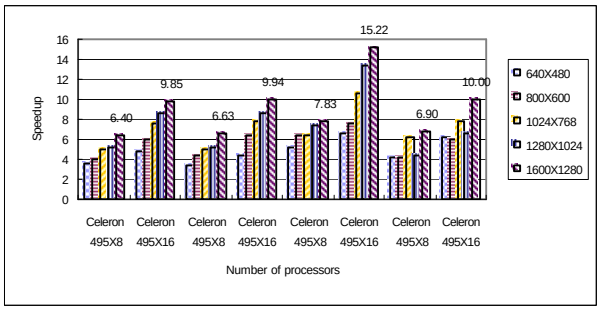

Figure 8: Speedups of PVMPOV diagrams

4 Conclusion and Future Work

In traditional scalable computing clusters, there is related high powered Intel or AMD based PC's associated with several gigabytes of hard disk space spread around. As far as a cluster administrator is concerned, the installer may have to install an operating system and related software for each cluster node. Every node (excluding the Server) is equipped without any hard disk and is booted from the network. Since it has booted successfully, it works as same as a fully equipped one. We will explain how to set up a diskless cluster for computing purpose. In this paper, a SMP-based PC cluster (16 processors), with diskless slave nodes, is proposed and built. In order to take advantage of a cluster system, we presented the basic programming techniques by using Linux/PVM to implement a PVM-based matrix multiplication program. Also, a real application PVMPOV by using parallel ray-tracing techniques was examined. The experimental results show that the highest speedups are obtained for matrix multiplication and PVMPOV,

when the total number of processors is 16, by creating 16 tasks on SMPs cluster. The results of this study will make theoretical and technical contributions to the design of a message passing program on a Linux SMP clusters.

References

1. R. Buyya, *High Performance Cluster Computing: System and Architectures*, Vol. 1, Prentice Hall PTR, NJ, 1999.
2. R. Buyya, *High Performance Cluster Computing: Programming and Applications*, Vol. 2, Prentice Hall PTR, NJ, 1999.
3. http://www.ltsp.org, *Linux Terminal Server Porject*.
4. http://www.lam-mpi.org, *LAM/MPI Parallel Computing*.
5. http://www.haveland.com/povbench, *POVBENCH – The Official Home Page*.
6. http://www.epm.ornl.gov/pvm/, *PVM – Parallel Virtual Machine*.
7. T. L. Sterling, J. Salmon, D. J. Backer, and D. F. Savarese, *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters*, 2nd Printing, MIT Press, Cambridge, Massachusetts, USA, 1999.
8. B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall PTR, NJ, 1999.
9. M. Wolfe, *High-Performance Compilers for Parallel Computing*, Addison-Wesley Publishing, NY, 1996.
10. C. T. Yang, S. S. Tseng, M. C. Hsiao, and S. H. Kao, "A Portable parallelizing compiler with loop partitioning," *Proc. of the NSC ROC(A)*, Vol. 23, No. 6, 1999, pp. 751-765.
11. Chao-Tung Yang, Shian-Shyong Tseng, Yun-Woei Fan, Ting-Ku Tsai, Ming-Hui Hsieh, and Cheng-Tien Wu, "Using Knowledge-based Systems for research on portable parallelizing compilers," Concurrency and Computation: Practice and Experience, vol. 13, pp. 181-208, 2001.