A Practical Guide to Building High-Performance Computing Clusters

Book · January 2011

DOI: 10.13140/2.1.1153.1529

Tsung-Lung Li
National Chia-Yi Universty
43 PUBLICATIONS **228** CITATIONS

A Practical Guide to Building High-Performance Computing Clusters

Tsung-Lung Li

Professor
Department of Electrophysics
University of Chia-Yi University
Chiayi
Taiwan

Dedicated to Arlene and Angela,
who waited for me patiently
while I spent many hours
on the cluster

Preface

The demand for computational power has significantly increased in the nanoscience era because simulations are usually performed at molecular levels with quantum mechanics. At the same time, the performance and availability of both commodity computers and inexpensive high-speed networking hardware have increased drastically in the past decades. Hence, scientists and engineers working on nanoscience simulations may attempt to build a small- or medium-size high-performance computing cluster at their laboratories. However, building a high-performance computing cluster is not a simple task, especially for nanoscience researchers whose specialties are not on subjects in computer science.

In this monograph, the procedures to build a high-performance computing cluster are presented in a way that is followable to less system-oriented researchers. The interested readers can follow the idea or can use it as a guide to build their own system.

Chiayi, *Tsung-Lung Li*
July 2011

Acknowledgment

The author would like to acknowledge two workshops provided by the National Center for High-Performance Computing (NCHC), Hsinchu, Taiwan, including "DRBL and Clonezilla Centralized Management Environment" and "Workshop on Integrated PC Clusters." This work is partly inspired by these workshops. The author would also like to thank the Free Software Laboratory, NCHC, and its team members, Jr-Huang Shiau, Chen-Kai Sun, Yao-Tsung Wang, and Yu-Chin Tsai for their devotion to the DRBL package, and their technical support on this project. The supports of the National Science Council (NSC) by Grant Nos. NSC-96-2112-M-415-003-MY3 and NSC-99-2112-M-415-008 and of the National Center for Theoretical Sciences (NCTS) are also acknowledged.

Contents

Chapter 1

Introduction

Simulations are usually performed at molecular levels for nanoscience problems. Frequently, the problems are solved quantum-mechanically and the solution schemes are computationally expensive[1, 2]. With the advent of the nanoscience era, the demand for computational power has increased significantly. At the same time, the performance and availability of commodity-off-the-shelf computers and inexpensive high-speed networking hardware have increased drastically in the past decade. Scientists and engineers working on nanoscience simulations may attempt to build a small- or medium-size high-performance computing cluster at their laboratories for parallelized applications.

However, building a high-performance computing cluster is not a simple task, especially for nanoscience researchers whose specialties are not on the subjects of computer sciences. With the usage of Diskless Remote Boot for Linux (DRBL), the construction and maintenance of a computing cluster are greatly simplified. Hence, building a computing cluster from scratch becomes feasible for researchers without much prior exposure to computer sciences.

In this work, the processes to build a high-performance computing cluster with diskless clients are presented. A computing cluster with diskless clients reduces the construction and maintenance efforts, and improves the cluster hardware reliability. The construction of a computing cluster involves the installation of several software systems such as networking, parallelization, queuing, and monitoring systems. These software systems have to work in harmony with the operation system and application software. Hence, in addition to the system hardware, the installation processes of the software systems mentioned above will be illustrated.

Of course, the processes to be presented in this chapter are not the only way to construct a high-performance computing cluster. There are many other good approaches[3, 4]. The materials in this chapter is just one of the feasible way to build a computing cluster. The path taken in this chapter can be followed by a less system-oriented researcher.

Not much complicated knowledge in computer science is assumed for the readers. However, some basic knowledge on the Linux operating system is required. The readers are assumed to have some experiences on the operation and administration of a single machine Linux. The installation and configuration steps to construct a computing cluster are given in such details that interested readers can follow the ideas or can use them as a guide to build their own systems.

Chapter 2

Hardware

This high-performance computing cluster to be constructed in this chapter consists of one server and three diskless client nodes. Each client obtains the image of the operating system from the server at boot time, and accesses file systems exported by the server after it is booted. The client operation relies on the networking between nodes. The hardware of the sever and client nodes is described as follows.

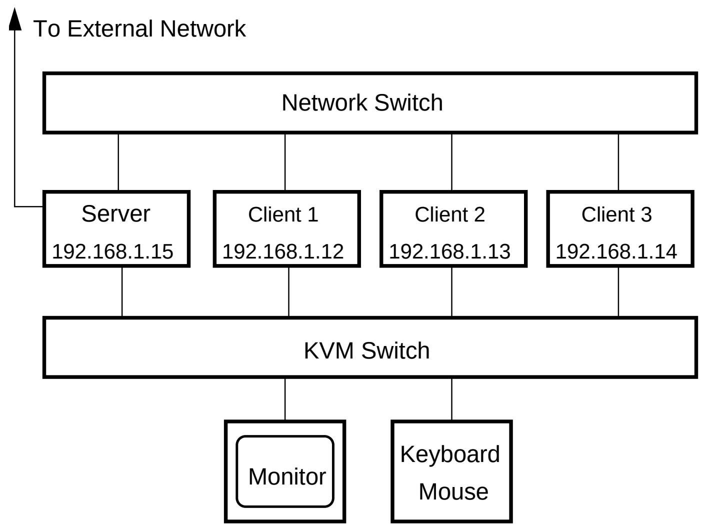


Figure 2.1: Illustration of the cluster hardware configuration.

Each client is equipped with dual Intel Quad-Core Xeon E5405 2.00GHz processors[5], a Tyan Tempest i5100W S5376G2NR motherboard[6], six banks of 2G DDR2-667 ECC registered SDRAM memory, and a 8GB SATA solid-state disk[1].

On-board with the motherboard are dual channels of Intel 82573V Gigabit Ethernet LAN controllers and one XGI Z9S video controller. The solid-state disk on each client is mainly to be used as local swap space for two reasons. First, the client will not crash due to lack of swap space if memory swapping ever occurs. Second, memory swapping on the local space reduces network loading between server and clients. No other disk is installed on each client for this implementation of high-performance computing cluster. Utilization of solid-state disks on the client nodes increases the cluster hardware reliability because there are no moving parts on the

[1]A total of 12 GB memory and 8 GB swap is insufficient for some of our applications. The solid-state disk was replaced later by a SATA hard drive to augment the swap space. It is suggested to install as many memories as the readers can afford. This will drastically improve the system performance. For the present mother board, each node can have up to 48GB memory.

drive. The on-board video controller is adequate for the client because only text display mode will be used for the client.

In addition to the hardware mentioned above, the server has three Seagate 500GB SATA hard disks, a DVD SATA drive, and an additional ASUS EAH4650 video controller. No solid-state disk is installed with the server because its swap space can conveniently located on one of the hard disks. The additional video card is required because the on-board video controller is not fast enough to support the 1680×1050 LCD monitor in the graphics display mode.

As shown in Fig. 2.1, the intra-cluster communications between the server and the three clients are switched by a PCI 16-port Gigabit Switching HUB FXG-16TX using one of the dual channels of the Intel gigabit network controller on each computer. The second channel of the network controller is used for communication with the external network. The server and the clients are also connected to an ATEN Master View Maxiport ACS-1216A KVM switch for convenient monitoring and operation of the cluster.

As a result, each node of the cluster has two quad-cores CPUs (a total of 8 CPU cores per node) and 12GB SDRAM memory. The cluster has the total hard disk space of 1.5TB for storage and a gigabit network communication interface between nodes. Besides the solid-state disk mainly reserved for local swap space, the client does not have any hard disk. Hence, the client nodes of this computing cluster are systemless or, to some extent, diskless.

By the time this monograph is published, most of the hardware mentioned above may be no longer available on the market. The above list of hardware only serves as a demonstration to the idea. Readers may have to design their system hardware based on the idea.

Chapter 3

System Software

The installation of system software on the computing cluster is an integration process of several software systems as illustrated in Fig. 3.1, including an operating system, a networking system, a parallelization system, a queuing system, and a monitoring system.

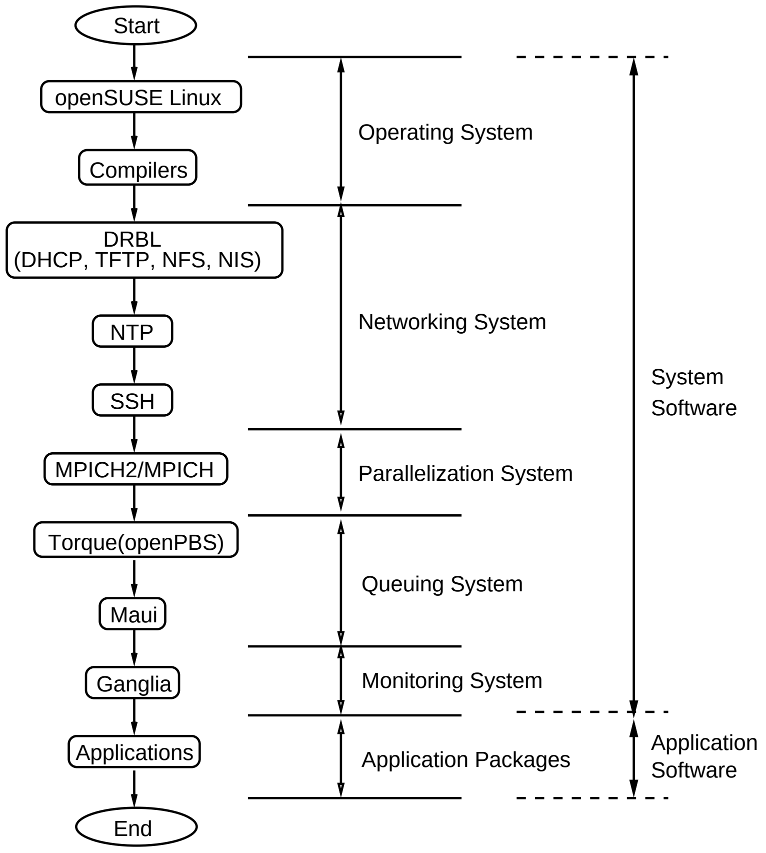


Figure 3.1: Installation process of system software and applications to the computing cluster.

The operating system of the high-performance computing cluster is based on the openSUSE 11.1 Linux[7] distribution. In addition to the GNU compilers[8], Intel[9] and Open64[10] compilers are installed. Clustering software systems include Diskless Remote Boot in Linux (DRBL)[11], SSH, NTP, Torque (openPBS)[12], Maui[13], and Ganglia[14]. The parallelization is achieved by MPICH2[15] and MPICH[16].

In the installation process, conflicting between the software systems has to be resolved. This is an important issue for the computing nodes of the cluster to co-operate properly. The installation processes of the pieces of system software mentioned above are given in the following subsections.

3.1 Linux System: openSUSE 11.1

The operating system of the computing cluster is based on openSUSE Linux 11.1 distribution[17]. Image `openSUSE-11.1-DVD-x86_64.iso` is installed to the server of the cluster[7]. The kernel version is 2.6.27.

The three hard disks are partitioned as follows. Device `/dev/sda` contains directories `/boot` (4GB), `/` (root, 40GB), `/opt` (40GB), `/tftpboot` (20GB), and `/home` (361GB). Device `/dev/sdb` contains directories `/tmp` (20GB) and `/work` (433GB), and `swap` (12GB). Device `/dev/sdc` contains directory `/srv` (465GB). The `ext3` file system is used for all partitions.

For the sake of completeness, all packages of the openSUSE Linux 11.1 are selected to be installed except for those in a few sections: `KDE3 Desktop Environment`, `KDE3 Base System`, `32-bit Runtime Environment`, `Laptop`, `TablePC`, and `Xen Virtual Machine Host Server`. However, the selection of the packages in section `KDE Development` turns back on the selection of two sections: `KDE3 Desktop Environment` and `KDE3 Base System`. Conflicting packages are selectively installed. For instance, packages `tftp` and `postfix` are favored over packages `aftp` and `sendmail`, respectively. Shell `bash` is the default for all users.

For networking, device `eth0` is utilized for external connection. The hostname is set to be sham.ncyu.edu.tw. IPv6 is enabled. "Traditional network setup with ifup" is selected for network connection. Fixed IP address 140.130.91.204, netmask 255.255.255.0, DNS, and gateway are set. Device `eth1` is used for internal connection. The hostname and fixed IP address associated with this device are sham-eth1.ncyu.edu.tw and 192.168.1.15, respectively. Notice that the IP address of `eth1` must be within the private IP address ranges, for instance, within 192.168.*.*. All other settings are the same as device `eth0`.

Depending on the hardware of the video card, the video driver might have to be updated to make better use of the video hardware. Since the ASUS video card added to the server uses ATI Radeon HD4650 chipset, the video driver is updated with the following steps. First, obtain the ATI video driver package[18], and issue as root

```
sh ati-driver-installer-9-12-x86.x86_64.run
```

and select to generate package `fglrx64_7_4_0_SUSE111-8.681-1.x86_64.rpm`. Then, make sure that there is no previously installed version of ATI video driver by

```
rpm -qa | grep fglrx64
```

Remove the older version by command "`rpm -e pacakge_name`" if there is any. Install the generated package and generate a new `/etc/X11/xorg.conf` file by commands

```
rpm -ihv fglrx64_7_4_0_SUSE111-8.681-1.x86_64.rpm
cp -a /etc/X11/xorg.conf /etc/X11/xorg.conf.orig
aticonfig --initial
```

The second command is to back up the original `/etc/X11/xorg.conf` file. It is the convention of this work to name the backup configuration files with a `.orig` extension. The new video driver takes effect after rebooting the system, and can be verified by clicking on YaST → Hardware → Graphics Card and Monitor. Throughout this chapter, notation "Menu → Submenu" means clicking on Menu and then on Submenu.

A few additional tunings on the operating system are suggested in this paragraph. First, configure `/etc/sudoers` by `visudo`. Second, activate `vsftp` and turn on run levels 3 and 5 by clicking on YaST → System → System Services (Runlevel). Configure `vsftp` by modifying the following lines in `/etc/vsftpd.conf`,

```
write_enable=YES
local_enable=YES
local_umask=022
ftpd_banner="Welcome to sham FTP service."
```

Third, add software repositories by

```
zypper addrepo \
       http://ftp.skynet.be/pub/packman/suse/11.1 packman
zypper addrepo \
       http://packman.iu-bremen.de/suse/11.1 \
       packman-mirror
```

Make sure that either `packman.repo` or `packman-mirror.repo` is enabled, not both. Fourth, install extra packages by clicking on YaST → Software → Software Management. In particular, packages `octave`, `rasmol`, `xfig`, `xmgrace`, and many others can be useful. Fifth, configure printer by clicking on YaST → Hardware → Printer. Sixth, apply online update at the end by clicking on YaST → Software → Online Update. The kernel is then updated to be version 2.6.27.39-0.2.

Reboot the system and check all changes on settings work as expected. The installation of openSUSE 11.1 Linux operating system is completed. The system is ready for the installation of clustering software.

3.2 Compilers: Intel and Open64

Compilers are an important component to a high-performance computing cluster. The GNU compilers[8] included in the openSUSE Linux 11.1 distribution, gcc 4.3.2 and gfortran 4.3.2 are installed. In addition to the GNU ones, Intel[9] and Open64[10] compilers are also installed to the computing cluster. Their installation processes are given in the following two subsections.

Intel compilers

Because library `libstdc++.so.5` is required for this version of Intel compilers, packages `libstdc++33`, `libstdc++33-devel`, and `libstdc++33-doc` are installed by clicking on YaST → Software → Software Management prior to installing the Intel compilers.

Intel Fortran (`ifort`) and C^{++} (`icc`) compilers version 11.1.046 are installed to the cluster server by using packages `l_cprof_p_11.1.046_intel64.tgz` and `l_cproc_p_11.1.046_intel64.tgz`, respectively[9]. These packages are installed to directory `/opt/intel/Compiler/11.1/046/`. Installed components include Intel Fortran and C^{++} compilers, Debugger, MKL, IPP, and TBB. Mathematical Kernel Library (MKL) offers highly optimized mathematical routines for high-performance applications. Adding the following lines to user's personal initialization file, `.bash_profile` to sets up the MKL environment at login shell.

```
source /opt/intel/Compiler/11.1/046/bin/ifortvars.sh \
       intel64
source /opt/intel/Compiler/11.1/046/bin/iccvars.sh intel64
```

Intel MKL Fortran 90 interfaces can be created by `makefile` at `/opt/intel/Compiler/11.1/046/mkl/interfaces/[blas95,lapack95]` as follows. The Fortran 90 interfaces for BLAS are created by

```
cd /opt/intel/Compiler/11.1/046/mkl/interfaces/blas95
make libem64t \
     INSTALL_DIR= \
     /opt/intel/Compiler/11.1/046/mkl/interfaces/blas95 \
     FC=ifort
```

Library and interface files are generated at `/opt/intel/Compiler/11.1/046/mkl/interfaces /blas95/[lib,include]/`. The Fortran 90 interfaces for LAPACK are created by

```
cd /opt/intel/Compiler/11.1/046/mkl/interfaces/lapack95
make libem64t \
     INSTALL_DIR= \
     /opt/intel/Compiler/11.1/046/mkl/interfaces/lapack95 \
     FC=ifort
```

Library and interface files are generated at `/opt/intel/Compiler/11.1/046/mkl/interfaces /lapack95/[lib,include]/`.

The library of FFTW3 Fortran wrappers to Intel MKL is built by

```
cd /opt/intel/Compiler/11.1/046/mkl/interfaces/fftw3xf
make libem64t compiler=intel
```

The library `/opt/intel/Compiler/11.1/046/mkl/interfaces/fftw3xf/libfftw3xf_intel.a` is created.

Documentations on the compilers and libraries are at `/opt/intel/Compiler/11.1/046/ Documentation/en_US/[documentation_f.htm,documentation_c.htm]`.

Open64 compilers

Open64 Fortran (`openf90`), C and C^{++} (`opencc` and `openCC`) compilers version 4.2.2.2.1 are installed to the cluster server by[10]

```
rpm --prefix=/opt/x86_open64/4.2.2.2.1 -ivh \
    x86_open64-4.2.2.2-1.x86_64.rpm
```

Open64 compilers are installed to directory `/opt/x86_open64/4.2.2.2.1/`. Add the following lines to user's `.bash_profile` file to include the directory in the search path.

```
PATH=$PATH:/opt/x86_open64/4.2.2.2.1/bin
export PATH
```

User's guide for Open64 compilers, `x86_open64_user_guide.pdf` is installed to directory `/opt/x86_open64/4.2.2.2.1/doc/` for future reference.

3.3 Diskless Remote Boot Linux (DRBL)

The installation of the networking system to the computing cluster is greatly simplified by using the Diskless Remote Boot Linux (DRBL) package[11, 19, 20, 21]. In the process of installing DRBL, several network services are installed, if not previously, and configured automatically, including the dynamic host configuration protocol (DHCP)[22], the trivial file transfer protocol (TFTP), the network file system (NFS), and the network information service (NIS). These services are required for the operation of a high-performance computing cluster with diskless clients.

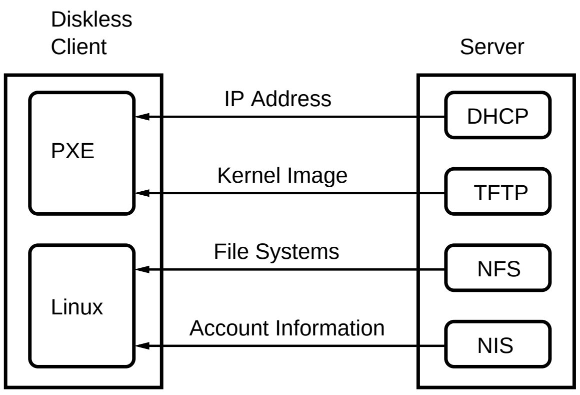


Figure 3.2: Boot process of the diskless client using DRBL.

At boot time, the client node uses preboot execution environment (PXE)[23] boot agent to initiate the booting process as demonstrated in Fig. 3.2. The PXE client first requests an Internet protocol (IP) address from the DHCP server, and then fetches the kernel image from the server node via the TFTP server[1]. After the client node is booted, the server file systems are exported to the client using NFS. User account information is distributed by NIS among cluster nodes. Hence, no kernel image and no file system are required to physically install on the client node. A high-performance computing cluster with diskless clients is thus achieved.

Notice that during the installation of the openSUSE Linux system, a hard disk partition on the server node, `/tftpboot` is created. This partition is reserved for the DRBL system. The installation steps of the DRBL system are itemized below.

1. Turn off that firewall temporarily by YaST → Security and Users → Firewall.

2. Check that "Traditional Method with ifup" mode is used for networking by YaST → Network Devices → Network Card → Global Options.

3. Check that the interface `eth1` is configured for internal network connection with a static private IP address by YaST → Network Devices → Network Settings.

4. Download, import, and check the golden key by the following commands.

   ```
   rpm -e --allmatches gpg-pubkey-d7e8df3a-44c6e1d6
   wget http://drbl.nchc.org.tw/GPG-KEY-DRBL
   rpm --import GPG-KEY-DRBL
   rpm -qa gpg-pubkey | grep -i D7E8DF3A
   ```

   The first command is to remove all existing keys of the DRBL package, and is not required if the package is being installed for the very first time. The last command is to check that the key is properly downloaded.

5. Download the DRBL package `drbl-1.9.5-42.i386.rpm`[11] and then install it by the following commands.

   ```
   /opt/drbl/sbin/drblsrv -u
   rpm -Uvh drbl-1.9.5-42.i386.rpm
   ```

[1]Recall that during the installation of the openSUSE Linux system, tftp is favored over aftp to resolve a package conflict. The reason for this choice becomes obvious at this point. DRBL uses tftp rather than aftp.

The first command is to uninstall DRBL if it is previously installed. The DRBL package installed on this cluster is version 1.9.5-42.

6. Install the DRBL server as root by

   ```
   /opt/drbl/sbin/drblsrv -i
   ```

7. Change the following lines in `/opt/drbl/conf/drbl.conf` for the server node to export its /work (read-write) and /srv (read-only) directories to clients.

   ```
   diskless_root_dir_ro_user_add="/srv"
   diskless_root_dir_rw_user_add="/work"
   ```

   On this computing cluster, directories /work and /srv are intended for temporary working space and long-term backup storage, respectively.

8. Add the following line to `/opt/drbl/conf/client-append-fstab` for each client to mount `/nodewk` on device `/dev/sda2` of their local solid-state disk after booting.

   ```
   /dev/sda1 swap         swap    defaults     0 0
   /dev/sda2 /nodewk      ext3    defaults     0 0
   ```

   On each client, two partitions, `/dev/sda1` (2GB) and `/dev/sda2` (6GB,ext3) are created on the SATA solid-state disk (8GB) with the following commands.

   ```
   fdisk /dev/sda
   mke2fs -c -t ext3 /dev/sda2
   mkswap -c /dev/sda1
   ```

9. Add the following lines to `/opt/drbl/conf/client-ip-hostname` for the clients to have pre-assigned node names.

   ```
   192.168.1.12   xeon12
   192.168.1.13   xeon13
   192.168.1.14   xeon14
   ```

   The names and IP addresses of the client nodes are listed in the file.

10. Add the following line to `/opt/drbl/conf/client-extra-service` for the clients to start NTP service automatically at boot.

    ```
    service_extra_added="ntp"
    ```

11. Run as a super user

    ```
    /opt/drbl/sbin/drblpush -i
    ```

    to set up DRBL. The NIS/YP domainname for this cluster is chosen to be xeoncluster. 2048 MBytes are allocated for swap on each client. Full DRBL mode and no clonzilla are selected. In the process, the MAC addresses of the clients are collected, and the contents are stored in file `/etc/drbl/macadr-eth1.txt`. After running this command, the firewall is turned back on automatically.

12. Change the firewall settings by clicking on YaST → Security and Users → Firewall → Select "Enable Firewall Automatic Starting" in the "Start-Up" section, and configure interfaces `eth0` and `eth1` to be "External Zone" and "Internal Zone," respectively, in the "Interfaces" section.

13. After installing or updating software, the following command can be used to update the clients if the settings saved in `/etc/drbl/drblpush.conf` are to be reused.

    ```
    /opt/drbl/sbin/drblpush -c /etc/drbl/drblpush.conf
    ```

A few tips on the usage of DRBL are given below.

1. All clients can be shut down using the following steps on the server: (1) Become root if starting with being a regular user; (2) Issue `/opt/drbl/sbin/dcs`; (3) Select item "All Select all clients" of the menu; (4) Select "shutdown - Shutdown client now." All clients are shut down and powered off properly after these steps.

2. The client mode can be restored to its first installation of this system by the following steps on the server: (1) Become root; (2) Issue `/opt/drbl/sbin/dcs`; (3) Select item "All Select all clients" of the menu; (4) Select "remote-linux-txt - Client_remote_Linux, _text_mode, _powerful_client."

3. The command `/opt/drbl/bin/drbl-doit` is useful for cluster administration. For example,

    ```
    /opt/drbl/bin/drbl-doit uptime
    sudo /opt/drbl/bin/drbl-doit -u root reboot
    sudo /opt/drbl/bin/drbl-doit -u root /sbin/poweroff
    sudo /opt/drbl/bin/drbl-doit -u root shutdown -h now
    ```

    The first command shows the uptime of each client. The second command reboots all clients. The last two commands shut down all clients. The following lines can be added to user's `~/.bash_profile` for the DRBL executables to be included in the search path.

    ```
    # Environment for DRBL
    PATH=$PATH:/opt/drbl/bin
    export PATH
    ```

The BIOS of the motherboard is AMIBIOS version V3.04. The PXE boots of the server and the clients have to be set differently by starting the BIOS setup utility to change the BIOS settings. The BIOS setup utility can be involved on each node at BIOS boot time. The PXE boot of the server is disabled by clicking on Chipset → South Bridge → Lan[1,2] Enabled → Lan[1,2] OP-ROM Disabled. On the contrary, the PXE boot of each client has to be enabled by clicking on Chipset → South Bridge → Lan[1,2] Enabled → Lan1 OP-ROM Enabled and Lan2 OP-ROM Disabled.

With these settings of DRBL and motherboard BIOS, the diskless clients can be booted after the server is properly booted. This concludes the installation of the DRBL package.

3.4 Network Time Protocol (NTP)

The network time protocol (NTP) synchronizes the system time of all nodes by connecting the cluster to an external network time server. NTP has to be configured properly for all cluster nodes to work in unison.

Package `ntp` is included with the openSUSE Linux 11.1 distribution, and is already installed during the installation process of the operating system. NTP is configured by clicking on YaST → Network Services → NTP Configuration. Select "Start NTP daemon: Now and On" and add a time server in the "General Settings" section. For this cluster, the time server `time.stdtime.gov.tw` is selected. In the "Security Settings" section, select "Run NTP Daemon in Chroot_Jail," and select "Open Port in Firewall" and then enable both network interfaces `eth0` and `eth1` through "Firewall Details."

After the above configuration step, a port will be opened on the firewall for the NTP daemon. This can be checked by clicking on YaST → Security and Users → Firewall → Interfaces. The NTP daemon, `xntpd` is added to the "External Zone."

The following line is added to `/etc/ntp.conf` after this administration step.

```
server time.stdtime.gov.tw iburst
```

Update the DRBL clients by command,

```
/opt/drbl/sbin/drblpush -c /etc/drbl/drblpush.conf
```

After this updating command, the `/etc/ntp.conf` file of each client also contains the above `server` line; the `/etc/ntp.conf` file of the server is modified by DRBL by appending a few lines starting with `restrict`. On the server, either remove the lines added to the `/etc/ntp.conf` file by DRBL or comment out the following line for the NTP to function properly.

```
#restrict default ignore
```

The NTP service is thus configured for temporal synchronization between all cluster nodes.

3.5 Secure Shell (SSH)

On this computing cluster, the secure shell (SSH) is used for logging into a cluster node and for executing commands on a cluster node. Properly configured SSH allows users to log into a cluster node from another without having to enter passwords. This trusted communication between nodes is a required status for a high-performance computing cluster.

SSH is installed with the installation process of the opensSUSE Linux 11.1 distribution. SSH can be activated using the system administration tool by clicking on YaST → System → System Service (Runlevel) → Enable sshd.

For a regular user, SSH is configured with the following steps.

1. Generate authorization keys by

   ```
   chmod 700 ~/.ssh
   cd ~/.ssh
   ssh-keygen
   chmod 644 ~/.ssh/id_rsa.pub
   cp -a ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
   ```

   Files `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub` are generated by sh-keygen.

2. The user remotely logs on each client from the server by

   ```
   ssh xeon[12-14]
   ```

   with password to add client informations to `~/.ssh/known_hosts` if it is the very first time for the user to log on. After these steps, the regular user can remotely log on a computing node from another without having to enter password.

For the super user, slightly more complicated steps are followed to configure SSH.

1. Log on as root and repeat the two steps by which a regular user uses to generate the authorization keys and client information. Steps same as a regular user are performed for the root user.

2. Copy the authorization keys and all other files in directory `/root/.ssh/` to each client by

   ```
   cd /root/.ssh
   for i in 12 13 14
   do
     cp -a authorized_keys id_rsa id_rsa.pub known_hosts \
           /tftpboot/nodes/192.168.1.$i/root/.ssh
   done
   ```

The last step is required on this cluster because the directory `/root` of each client is actually exported from the directory `/tftpboot/nodes/192.168.1.$i/root/` of the server in the DRBL construct. After these steps, the root user can remotely log on each computing node from another without having to enter password.

3.6 Message Passing Interface: MPICH2

The message passing interface (MPI) is used for parallelized applications[24, 25, 26]. In particular, MPI Chameleon (MPICH2[15] and MPICH[16]) implementations are installed on this computing cluster. MPICH2 is a newer implementation than MPICH. In contrast to MPICH, a multi-purpose daemon of MPICH2 has to be started on all clients participating in a parallel computation. The ring of daemons is used for communication between computing nodes in the MPICH2 implementation.

MPICH and MPICH2 will be installed from scratch using three different compilers on this cluster. Before the installation, notice that, in addition to MPICH, openSUSE 11.1 has other flavors of MPI bundled with the distribution. These have to be removed to avoid confusion. After backing up the factory-installed MPICH by the command,

```
cp -a /opt/mpich /opt/mpich.opensuse.orig
```

packages `mpich-devel-1.2.7p1-112`, `mpich-1.2.7p1-112`, `lam-devel-7.1.4-11`, `lam-7.1.4-11`, `openmpi-devel-1.2.2-19`, `openmpi-1.2.2-19` `mvapich2-devel-0.9.8-26`, `mvapich2-0.9.8-26`, `pvm-devel-3.4.5-89`, `pvm-3.4.5-89`, and `mpi-selector-1.0.0-17` are all removed by clicking on YaST → Software → Software Management if they are ever installed.

MPICH2 version 1.2.1 and MPICH version 1.2.7p1 (release 2005/11/04) will be compiled by Intel, Open64, and GNU compilers, and installed in separate directories on this computing cluster. The installation processes of MPICH2 version 1.2.1 will be first illustrated in this subsections; while those of MPICH version 1.2.7p1 will presented in the next subsection. MPICH is installed on this computer cluster only for backward compatibility. Usage of MPICH2 is encouraged over MPICH because the development of MPICH is no longer continued.

MPICH2 compiled by Intel compilers

MPICH2 version 1.2.1 package `mpich2-1.2.1.tar.gz` is compiled with Intel compilers version 11.1.046 and installed with the following steps.

1. Unpack the package in a work directory by command,

   ```
   tar -xvzpf mpich2-1.2.1.tar.gz
   ```

2. Configure and build the package in the work directory by

   ```
   CC=icc
   CXX=icc
   F77=ifort
   F90=ifort
   export CC CXX F77 F90
   ./configure \
       --prefix=/opt/mpich2/1.2.1/intel-11.1.046 \
       --with-pm=mpd --with-device=ch3:ssm \
       | tee configure.log
   make | tee make.log
   make install | tee install.log
   ```

   The MPICH2 package compiled by the Intel compilers is installed to directory `/opt/mpich2 /1.2.1/intel-11.1.046`. Documentation files in HTML format start with `/opt/mpich2/ 1.2.1/intel-11.1.046/www/index.htm`. The configuration and make log files generated in this step are copied to directory `/opt/mpich2/1.2.1/intel-11.1.046/log/` for future reference.

3. Add the following lines to user's `~/.bash_profile` for the executables and documentations to be accessible.

   ```
   PATH=/opt/mpich2/1.2.1/intel-11.1.046/bin:$PATH
   export PATH
   MANPATH= \
     /opt/mpich2/1.2.1/intel-11.1.046/share/man:$MANPATH
   export MANPATH
   ```

4. Perform the following tests.

   ```
   which mpd
   which mpicc
   which mpiexec
   which mpirun
   ```

   To avoid confusion, the PATH setting in `~/.bash_profile` pointing to MPICH version 1.2.7p1 can be disabled first, or set the search path of MPICH2 before MPICH. MPICH2 and MPICH both have commands with common names: `mpirun`, `mpicc`, `mpicxx`, `mpif77`, and `mpif90`. This can be the source of confusion.

5. Create file `~/.mpd.conf` with the following line, and set the file protection mode to be 600.

```
secretword=quantum-xeoncluster
```

6. Create file `mpd.hosts` with the following contents

   ```
   xeon12
   xeon13
   xeon14
   ```

   This file is saved as `/etc/mpd.hosts` with the protection mode of 644. This step is not necessary; it is done only for convenience of usage.

7. Start the ring of daemons by the following command and options,

   ```
   mpdboot -n 4 -f /etc/mpd.hosts --ncpus=8 \
           --ifhn=sham-eth1
   ```

   This command sets up a daemon ring of 4 machines: 1 server and 3 client nodes. Option `-n` is the number of nodes to start, and it is 4 because there are 4 nodes on this computing cluster. Option `--ncpus=8` is used because there are eight CPU cores on each node. Option `--ifhn=sham-eth1` has to be used because there are two network interfaces on each cluster node. The one for cluster internal communication is interface sham-eth1. A partial list of /etc/hosts is as follows.

   ```
   127.0.0.1       localhost
   127.0.0.2       sham.ncyu.edu.tw sham
   140.130.91.204  sham.ncyu.edu.tw sham
   192.168.1.15 sham-eth1
   ```

   `sham-eth1` and `sham` refer to the external and internal network interfaces, respectively. The booted ring of daemons can be checked by command

   ```
   mpdtrace -l
   ```

   The ring of daemons can be terminated by command

   ```
   mpdallexit
   ```

8. Do a simple test on the ring with

   ```
   mpiexec -n 24 hostname
   ```

   Option `-n` is the number of processes to be use for a computation. This command always involves the server in the computation job. If the server is to be excluded from the computation job, the following command can be used,

   ```
   mpiexec -machinefile /etc/mpd.mach -n 24 hostname
   ```

   `/etc/mpd.mach` is a user-created file with the following contents

   ```
   xeon12:8
   xeon13:8
   xeon14:8
   ```

This file does not have to be placed in `/etc`, it is there with the file protection mode of 644 just for the convenience of usage. The number of processes of 24 following the option `-n` is the maximal value for this system. Values over 24 will not run on this cluster.

9. The documentations of MPICH2 version 1.2.1 in the HTML format is located at `/opt/mpich2/1.2.1/intel-11.1.046/share/doc/index.htm`

10. Compile the example in `/opt/mpich2/1.2.1/intel-11.1.046/share/examples_logging /cpi.c` by command

    ```
    mpicc -o cpi cpi.c
    ```

    after copying it a a work directory. Run the compiled binaries by command

    ```
    mpiexec -machinefile /etc/mpd.mach -np 24 cpi
    ```

    or

    ```
    mpiexec -np 24 cpi
    ```

    The first command will not involve the server node in the computation; while the second will. Other examples in directories `examples_collchk`, `examples_graphics`, and `examples_logging` can be compiled and tested with similar methods.

11. For future reference, save the unpacked source tree of `mpich2-1.2.1.tar.gz` to `/opt/mpich2/1.2.1/source/`.

12. Create an executable `~/bin/mpich2-mpdboot` with the following contents.

    ```
    mpdboot -n 4 -f /etc/mpd.hosts --ncpus=8 \
            --ifhn=sham-eth1
    ```

    Then set the protection mode of the file by

    ```
    chmod 755 ~/bin/mpich2-mpdboot
    ```

    Add the following line to `~/.bash_profile`

    ```
    # Add path to local executables
    setenv PATH ${HOME}/bin:${PATH}
    ```

    so that the local executables are on the search path. With these settings, the ring of MPICH2 daemons can be started easily with the command `mpich2-mpdboot`.

MPICH2 compiled by Open64 compilers

MPICH2 version 1.2.1 package `mpich2-1.2.1.tar.gz` is compiled with Open64 compilers version 4.2.2.2 and installed with the following steps.

1. Unpack the package `mpich2-1.2.1.tar.gz` to a work directory.
2. Configure and build the package in the work directory by

   ```
   CC=opencc
   CXX=openCC
   FC=openf90
   F90=openf90
   export CC CXX FC F90
   ./configure \
       --prefix=/opt/mpich2/1.2.1/open64-4.2.2.2 \
       --with-pm=mpd --with-device=ch3:ssm \
       --disable-cxx | tee configure.log
   make | tee make.log
   make install | tee install.log
   ```

   The MPICH2 package compiled by Open64 compilers is installed to directory `/opt/mpich2/1.2.1/open64-4.2.2.2`. The configuration and make log files generated in this installation step are copied to directory `/opt/mpich2/1.2.1/open64-4.2.2.2/log/` for future reference.
3. Add the following lines to user's `~/.bash_profile` for the executables an manual pages to be on the search path.

   ```
   PATH=/opt/mpich2/1.2.1/open64-4.2.2.2/bin:$PATH
   export PATH
   MANPATH=/opt/mpich2/1.2.1/open64-4.2.2.2/man:$MANPATH
   export MANPATH
   ```
4. Perform the following tests.

   ```
   which mpd
   which mpicc
   which mpiexec
   which mpirun
   ```

   To avoid confusion, the PATH setting in `~/.bash_profile` pointing to MPICH version 1.2.7p1 can be disabled first, or set the search path of MPICH2 before MPICH.
5. Create file `~/.mpd.conf` with the following line, and set the file protection mode to be 600.

   ```
   secretword=quantum-xeoncluster
   ```
6. Create file `mpd.hosts` with the following contents

   ```
   xeon12
   xeon13
   xeon14
   ```

This file is saved in `/etc/mpd.hosts` with the protection mode of 644. This step is not necessary; it is done only for convenience of usage.

7. Start the ring of daemons by the following command and options,

   ```
   mpdboot -n 4 -f /etc/mpd.hosts --ncpus=8 \
           --ifhn=sham-eth1
   ```

   This command sets up a daemon ring of 4 machines: 1 server and 3 client nodes. For the meanings of the options of command `mpdboot`, refer to Subsection 3.6

8. Perform tests with methods similar to Subsection 3.6

MPICH2 compiled by GNU compilers

MPICH2 version 1.2.1 package `mpich2-1.2.1.tar.gz` is compiled with GNU compilers version 4.3.2 and installed with the following steps.

1. Unpack the package `mpich2-1.2.1.tar.gz` to a work directory.

2. Configure and build the package by

   ```
   CC=gcc
   CXX=gcc
   F77=gfortran
   F90=gfortran
   export CC CXX F77 F90
   ./configure --prefix=/opt/mpich2/1.2.1/gnu-4.3.2 \
               --with-pm=mpd --with-device=ch3:ssm \
               --disable-cxx | tee configure.log
   make | tee make.log
   make install | tee install.log
   ```

   The MPICH2 package compiled by the GNU compilers is installed to directory `/opt/mpich2/1.2.1/gnu-4.3.2`. The configuration and make log files generated in this step are copied to `/opt/mpich2/1.2.1/gnu-4.3.2/log/` for future reference.

3. The rest of the installation and testing steps are very similar to Subsection 3.6

3.7 Message Passing Interface: MPICH

In this subsection, the compilation, installation, and configuration of MPICH version 1.2.7p1 (release 2005/11/04) will be illustrated.

Before the installation, other flavors of MPI packages bundled with the openSUSE 11.1 distribution have to removed with methods given in the second paragraph of Subsection 3.6 to avoid possible confusion.

MPICH version 1.2.7p1 will be compiled by Intel, Open64, and GNU compilers, and installed to different directories. The installation processes will be presented in the following subsections.

MPICH compiled by Intel compilers

The MPICH version 1.2.7p1 package, `mpich.tar.gz` is compiled by the Intel compilers version 11.1.046 and is installed to the computing cluster with the following steps.

1. Unpack the package to a work directory, `/work/mpich` by

   ```
   tar -xvzpf mpich.tar.gz
   ```

2. Configure and build the package by

   ```
   cd /work/mpich
   CC=icc
   CXX=icc
   FC=ifort
   F90=ifort
   export CC CXX FC F90
   ./configure \
       --prefix=/opt/mpich/1.2.7p1/intel-11.1.046 \
       --with-device=ch_p4 --with-comm=sharedc \
       -rsh=ssh | tee configure.log
   make | tee make.log
   ```

3. The machine file at `/work/mpich/util/machines/machines.LINUX` is modified to be as follows.

   ```
   xeon12:8
   xeon13:8
   xeon14:8
   ```

4. Test the networking between nodes by

   ```
   /work/mpich/bin/tstmachines -v
   ```

5. Install the package by

   ```
   su -
   cd /work/mpich
   make install
   ```

   After this step, the MPICH package version 1.2.7p1 compiled by the Intel compilers is then installed to the directory `/opt/mpich/1.2.7p1/intel-11.1.046/`.

6. Compile C and Fortran testing codes by

   ```
   /opt/mpich/1.2.7p1/intel/bin/mpicc -o cpi cpi.c
   /opt/mpich/1.2.7p1/intel/bin/mpif90 \
                               -o pi3f90 pi3f90.f90
   ```

7. Run the testing codes by, for example,

   ```
   /opt/mpich/1.2.7p1/intel/bin/mpirun -nolocal -np 8 cpi
   ```

Option `-nolocal` excludes the server from the computation job. Option `-np` specifies the number of processors to be involved in the MPI run. For the present hardware, the maximal number that can be set for option `-np` is 24, in which case all of the client processors on this cluster are involved with the MPI run.

8. Add the following lines to user's `~/.bash_profile` for the MPICH executables and documentations to be on the user's search path.

   ```
   # Environment for MPICH 1.2.7p1 compiled by
   # Intel compilers version 11.1.046
   PATH=/opt/mpich/1.2.7p1/intel-11.1.046/bin:$PATH
   export PATH
   MANPATH=/opt/mpich/1.2.7p1/intel-11.1.046/man:$MANPATH
   export MANPATH
   ```

   For MPICH documentations in the HTML format, use a browser to read `/opt/mpich/1.2.7p1/intel-11.1.046/www/index.html`.

Three categories of testing codes are provided by MPICH version 1.2.7p1. The testing steps and results of the first two categories are given below. The third category is performance testing. Its testing steps and results will be presented in Subsection 3.7.

First, several testing codes in C, FORTRAN, and Fortran 90 are included in the directory `/work/mpich/mpich-1.2.7p1/examples/basic/`. They can be compiled and tested with similar methods mentioned above. Codes `cpi.c`, `srtest.c`, `systest.c`, `unsafe.c`, `fpi.f`, and `pi3f90.f90` compile and run successfully.

Second, codes in `/work/mpich/mpich-1.2.7p1/examples/test/` are tested by steps as follows. A line in `configure` of the above directory is changed to be

```
MPIRUNARGS='"-nolocal"'
```

The option `-nolocal` is needed for this cluster. Use the following commands to perform the test.

```
cd /work/mpich/mpich-1.2.7p1/examples/test/
./configure \
     -mpichpath=/opt/mpich/1.2.7p1/intel-11.1.046/bin
make testing | tee testing.log
```

Results reported by the coeds show that the tests are successful.

MPICH compiled by Open64 compilers

The MPICH version 1.2.7p1 package, `mpich.tar.gz` is compiled by Open64 compilers version 4.2.2.2 and is installed to the computing cluster using the following steps.

1. Unpack the package to a work directory, say `/work/mpich`.

2. Configure and build the package by

   ```
   cd /work/mpich
   CC=opencc
   CXX=openCC
   FC=openf90
   F90=openf90
   ```

```
export CC CXX FC F90
./configure \
      --prefix=/opt/mpich/1.2.7p1/open64-4.2.2.2 \
      --with-device=ch_p4 --with-comm=shared \
      -rsh=ssh | tee configure.log
make | tee make.log
```

3. The machine file at `/work/mpich/util/machines/machines.LINUX` is modified to be as follows.

```
xeon12:8
xeon13:8
xeon14:8
```

4. Test the network connection between nodes by

```
/work/mpich/bin/tstmachines -v
```

5. Install the package by

```
cd /work/mpich/
make install
```

   MPICH version 1.2.7p1 compiled by the Open64 compilers is then installed to the directory `/opt/mpich/1.2.7p1/open64-4.2.2.2/`.

6. Add the following lines to user's `~/.bash_profile` for the MPICH executables and documentations on the the search path.

```
# Environment for MPICH 1.2.7p1 compiled by
# X86 Open64 compilers version 4.2.2.2
PATH=/opt/mpich/1.2.7p1/open64-4.2.2.2/bin:$PATH
export PATH
MANPATH=/opt/mpich/1.2.7p1/open64-4.2.2.2/man:$MANPATH
export MANPATH
```

The testing steps for the three categories of testing codes provided by MPICH version 1.2.7p1 are the same as in Subsection 3.7. Several testing C and Fortran 90 codes in `/work/mpich/mpich-1.2.7p1/examples/basic/` compile and run successfully, including `cpi.c`, `srtest.c`, `systest.c`, `unsafe.c`, and `pi3f90.f90`.

MPICH compiled by GNU compilers

The MPICH version 1.2.7p1 package, `mpich.tar.gz` is compiled by GNU compilers version 4.3.2 and is installed to the computing cluster using the following steps.

1. Unpack the package to a work directory, say `/work/mpich`.

2. Configure and build the package by

```
CC=gcc
CXX=gcc
FC=gfortran
F90=gfortran
export CC CXX FC F90
./configure --prefix=/opt/mpich/1.2.7p1/gnu-4.3.2 \
            --with-device=ch_p4 --with-comm=shared \
            -rsh=ssh | tee configure.log
make | tee make.log
```

Notice that `mpif77` and `mpif90` are not created in `/work/mpich/bin` because `gfortran` does not support Fortran 90.

3. The machine file at `/work/mpich/util/machines/machines.LINUX` is modified to be as follows.

```
xeon12:8
xeon13:8
xeon14:8
```

4. Test the network connection between nodes by

```
/work/mpich/bin/tstmachines -v
```

5. Install the package compiled by the GNU compilers by

```
cd /work/mpich/
make install
```

MPICH version 1.2.7p1 is installed to `/opt/mpich/1.2.7p1/gnu-4.3.2/`.

6. Add the following lines to user's `~/.bash_profile` for the MPICH binaries and documentations to be on the user's search path.

```
# Environment for MPICH 1.2.7p1 compiled by
# GNU compilers version 4.3.2
PATH=/opt/mpich/1.2.7p1/gnu-4.3.2/bin:$PATH
export PATH
MANPATH=/opt/mpich/1.2.7p1/gnu-4.3.2/man:$MANPATH
export MANPATH
```

The testing steps for the three categories of testing codes provided by MPICH version 1.2.7p1 are the same as in Subsection 3.7. Some testing C codes in `/work/mpich/mpich-1.2.7p1/examples/basic/` compile and run successfully, including `cpi.c`, `srtest.c`, `systest.c`, and `unsafe.c`.

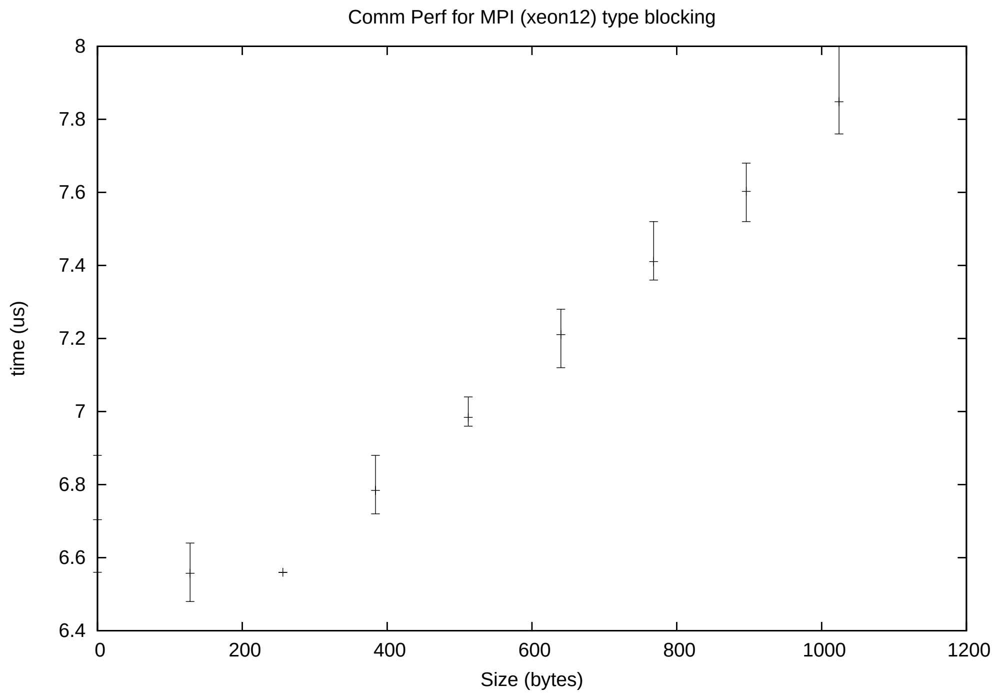


Figure 3.3: MPICH compiled by Intel compilers: type blocking (pt2ptshort).

Performance of MPICH

The codes in directory `/work/mpich/mpich-1.2.7p1/examples/perftest/` are for performance testing. They are compiled and tested as follows.

Configure the performance testing codes with command

```
cd /work/mpich/mpich/1.2.7p1/examples/test/
./configure --with-mpich
```

Change the maximum run time of `mpptest.c` by modifying the following line to be

```
static double max_run_time = 150.0*60.0;
```

Change the following line in `runmpptest` to be

```
set MPIRUNOPT = "-nolocal"
```

Change the following lines in `rungoptest` to be

```
set MPIRUNOPT = "-nolocal"
set MAXNP = 24
```

Then make the binaries by

```
make
```

Based on the suggestion in `README`, create and run the testing script `Run_all` with the following contents

```
#! /bin/sh
./runmpptest -short -pair -blocking -givedy -gnuplot \
             -fname pt2ptshort.mpl
./runmpptest -long -pair -blocking -givedy -gnuplot \
```

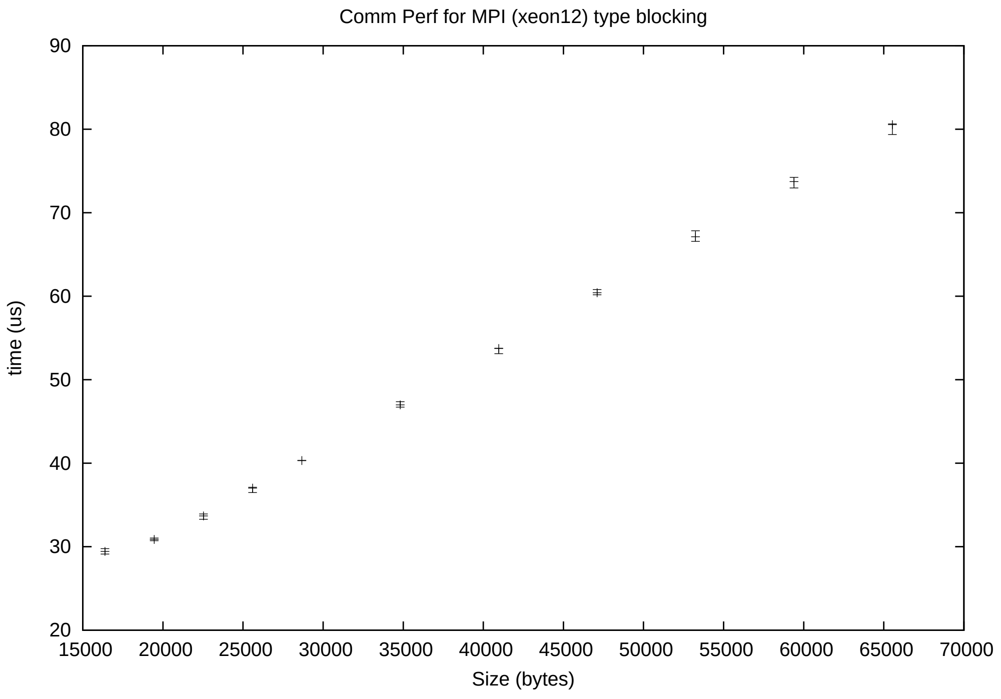


Figure 3.4: MPICH compiled by Intel compilers: type blocking (pt2ptlong).

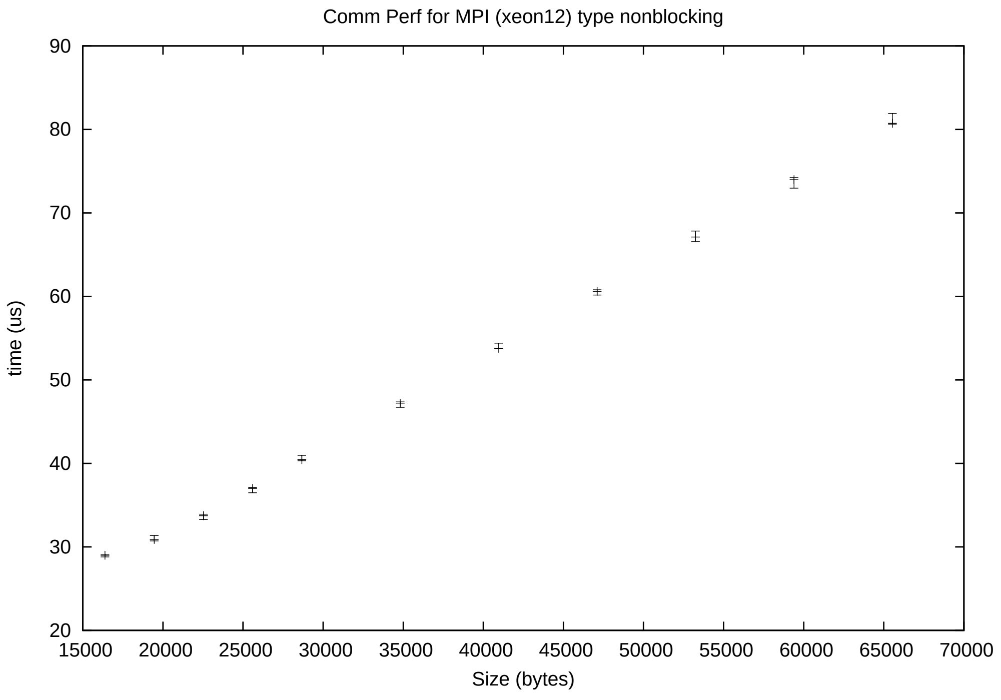


Figure 3.5: MPICH compiled by Intel compilers: type non-blocking (npt2ptlong).

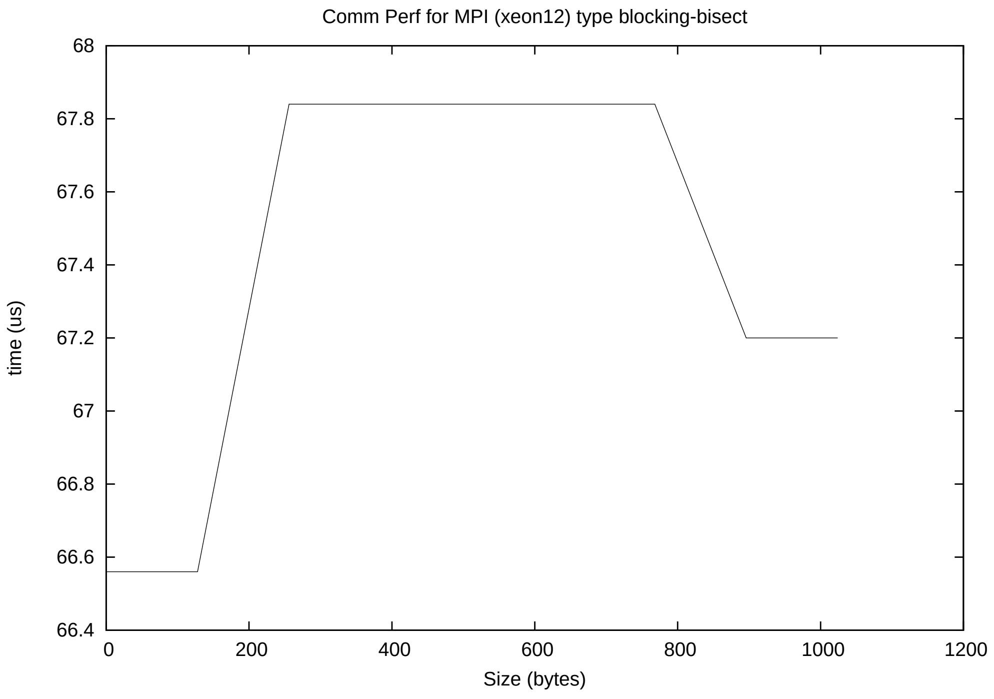


Figure 3.6: MPICH compiled by Intel compilers: type blocking bisection (bisectshort).

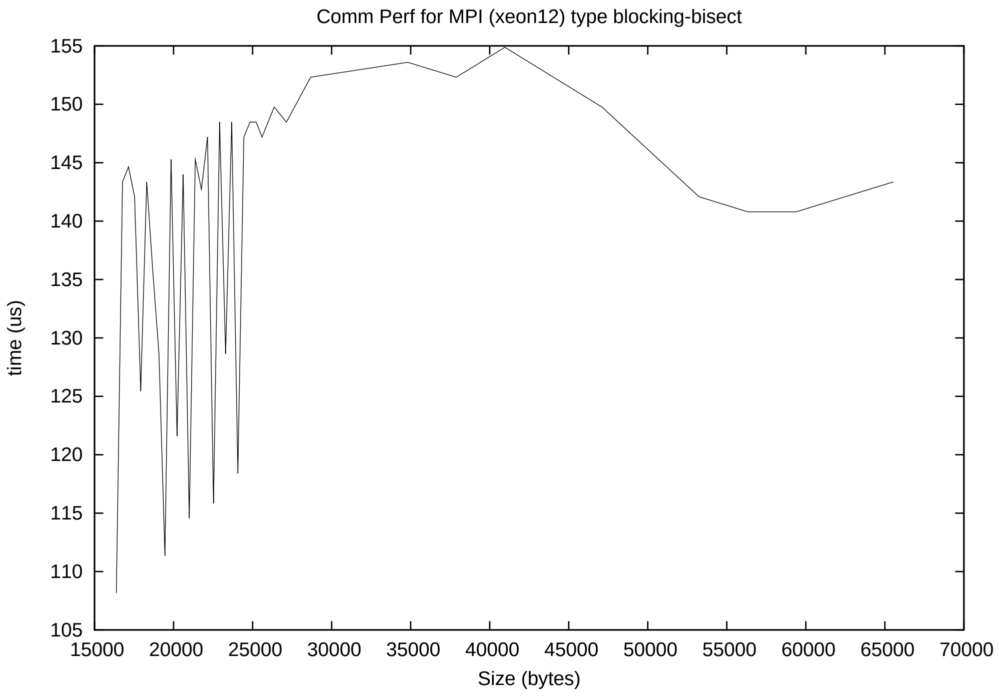


Figure 3.7: MPICH compiled by Intel compilers: type blocking bisection (bisectlong).

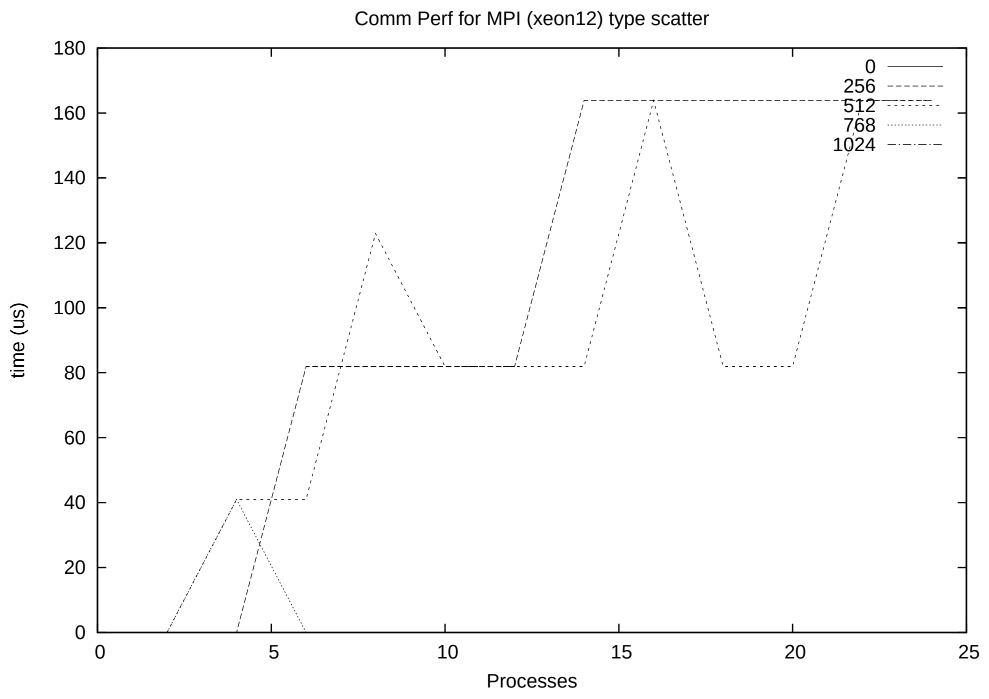


Figure 3.8: MPICH compiled by Intel compilers: type scatter (bcast).

.

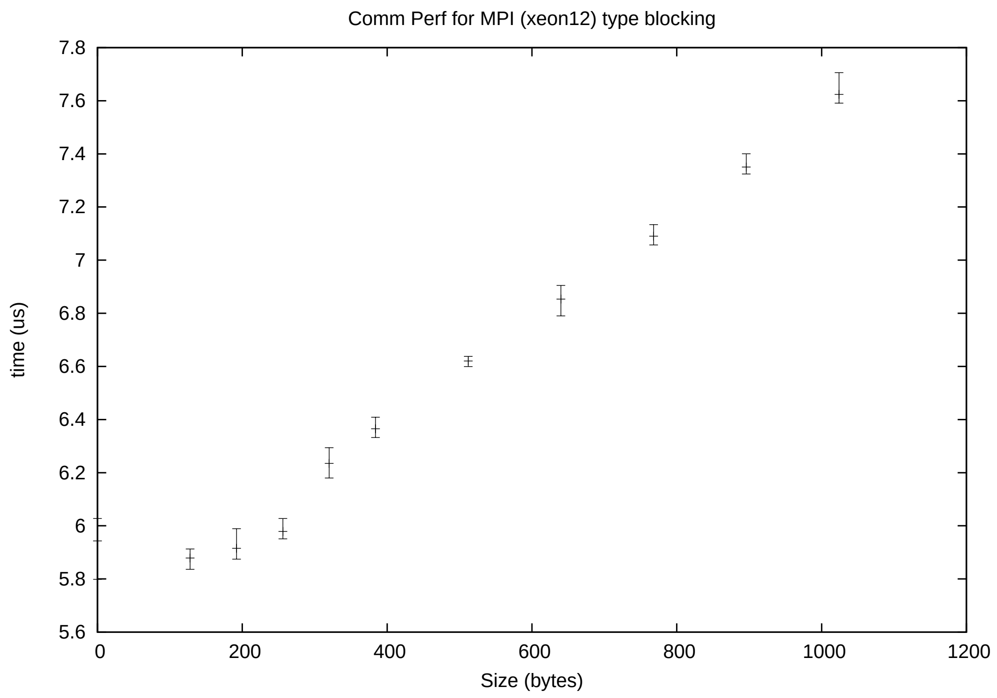


Figure 3.9: MPICH compiled by Open64 compilers: type blocking (pt2ptshort).

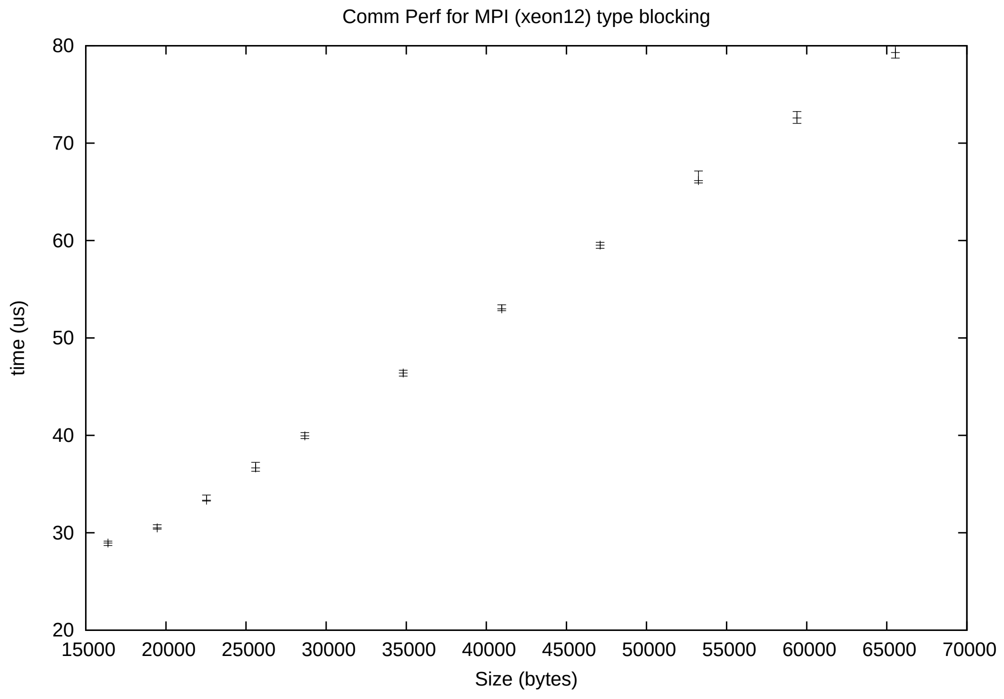


Figure 3.10: MPICH compiled by Open64 compilers: type blocking (pt2ptlong).

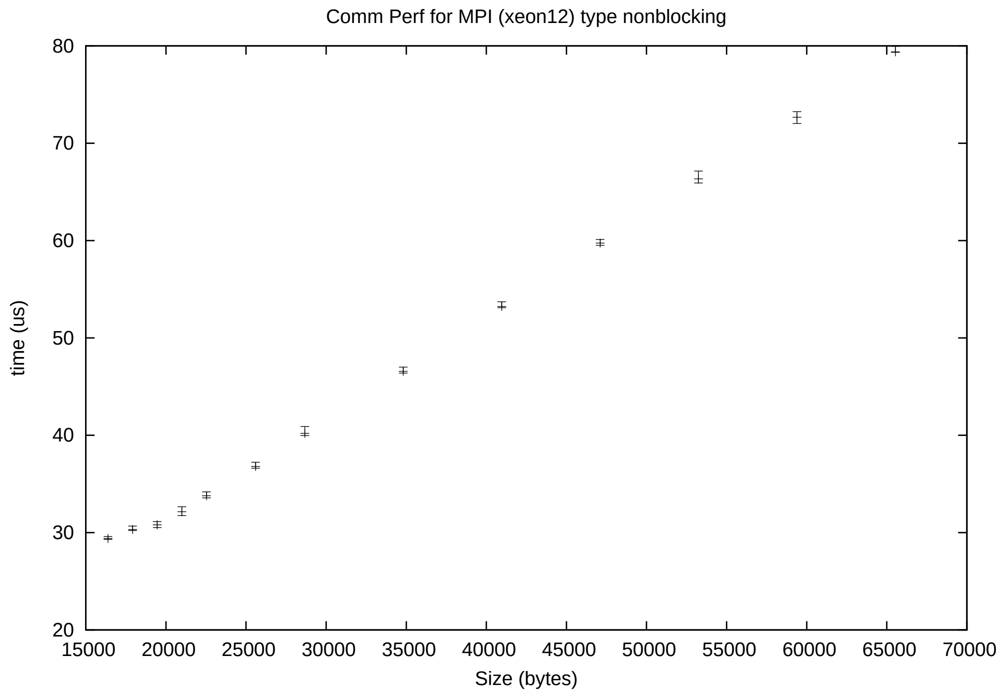


Figure 3.11: MPICH compiled by Open64 compilers: type non-blocking (npt2ptlong).

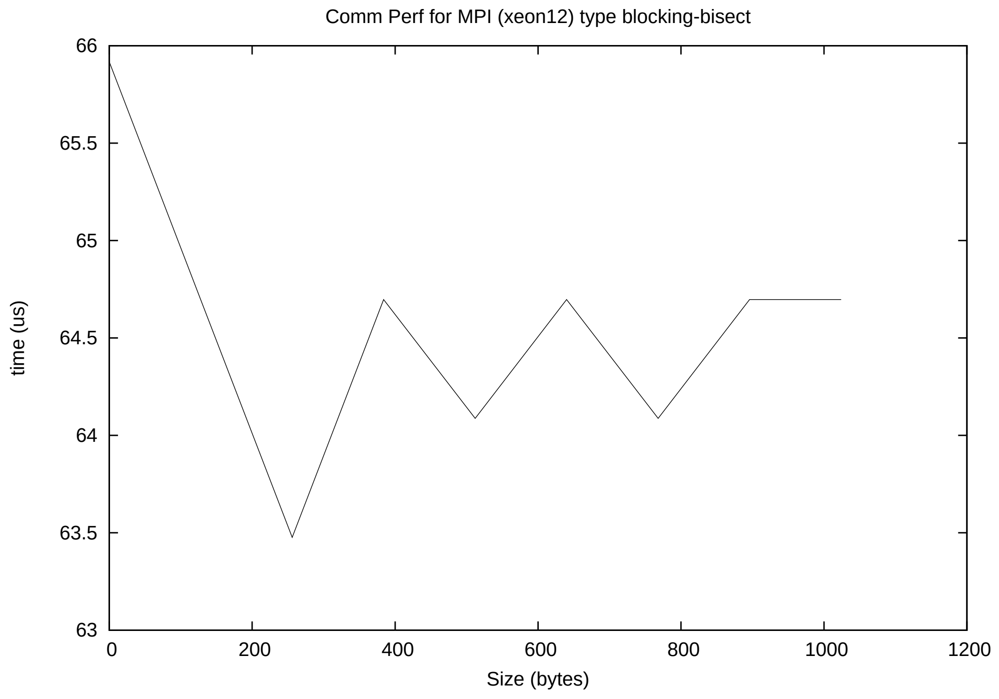


Figure 3.12: MPICH compiled by Open64 compilers: type blocking bisection (bisectshort).

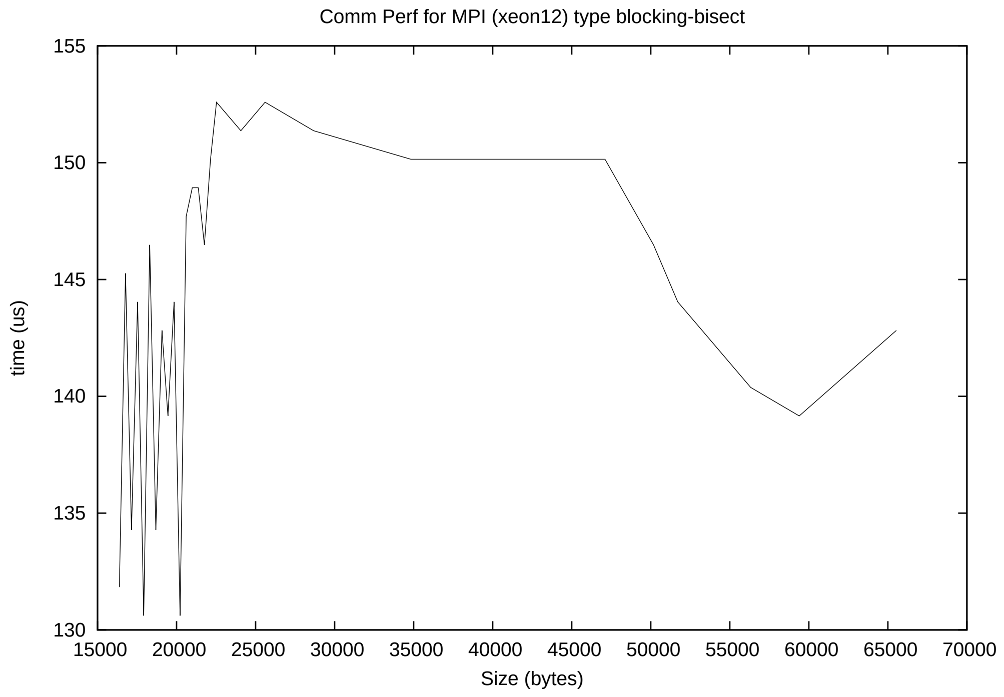


Figure 3.13: MPICH compiled by Open64 compilers: type blocking bisection (bisectlong).

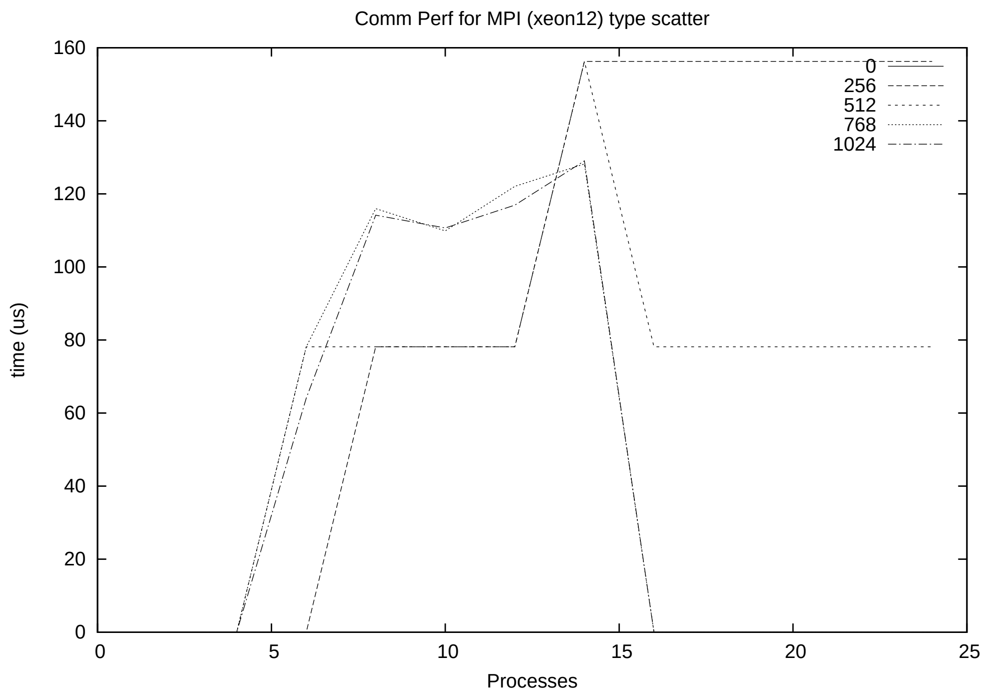


Figure 3.14: MPICH compiled by Open64 compilers: type scatter (bcast).

```
              -fname pt2ptlong.mpl
./runmpptest -long -pair -nonblocking -givedy -gnuplot \
              -fname nbpt2ptlong.mpl
./runmpptest -np 24 -bisect -short -blocking -gnuplot \
              -fname bisectshort.mpl
./runmpptest -np 24 -bisect -long -blocking -gnuplot \
              -fname bisectlong.mpl
./rungoptest -maxnp 24 -add -bcast -gnuplot \
              -fname bcast.mpl
```

The communication performances of MPICH compiled by the Intel compilers obtained by codes `pt2ptshort`, `pt2ptlong`, `nbpt2ptlong`, `bisectshort`, `bisectlong`, and `bcast` are shown in Figs. 3.3, 3.4, 3.5, 3.6, 3.7, and 3.8, respectively. The communication performances of MPICH compiled by the Open64 compilers are plotted in Figs. 3.9 to 3.13. Comparison of the performances of MPICH compiled the Intel and Open64 compilers shows that their performances are hardly distinguishable.

3.8 Queuing System: Torque (openPBS)

For high-performance computing clusters, a queuing system is an important component for efficient usage of system resources. The portable batch system, Torque (openPBS) is installed to the cluster[12]. Torque (openPBS) consists of three daemons: a job server, a job scheduler, and a job executor. The job server is responsible for services such as receiving and creating batch jobs, modifying and protecting the jobs, and running the job. The job scheduler controls the scheduling policies on the batch jobs. The job executor on the client is a machine-oriented mini-server (MOM server) that places the job into execution and returns the results to the server as it receives commands from the server. The queuing server and the scheduler run on the server of the computing cluster only; while the MOM server runs on every client node.

Installation

On this computing cluster, Torque (openPBS) is installed using package `torque-2.4.5.tar.gz`. Queuing properties are configured after the queuing system is started. The steps for the installation and configuration of the queuing system are listed below.

1. Unpack the downloaded source at `/opt/torque/2.4.5/source/`.

2. Configure, make, and install with the following commands.

   ```
   cd /opt/torque/2.4.5/source/torque-2.4.5
   ./configure --prefix=/opt/torque/2.4.5 \
               --with-server-home=/var/spool/torque \
               --with-default-server=sham \
               --with-sched=c --with-sched-code=fifo
   make
   make install
   ```

3. Create the node description file `/var/spool/torque/server_priv/nodes`. The `pbs_server` communicates with each of the MOMs running on clients. The contents of the file are as follows.

   ```
   xeon12 np=8
   xeon13 np=8
   xeon14 np=8
   ```

   Using this node description file, `pbs_server` communicate with the clients on which `pbs_mom` is up.

4. Create the server description file `/var/spool/torque/server_priv/serv er_name`. This file specifies the name of PBS server. The contents of the file are as follows.

   ```
   sham
   #sham-eth1
   ```

   The line starting with `#` is a comment.

5. Create the machine oriented mini-server (MOM) configuration file `/var/spool/torque/mom_priv/config`. with the following contents.

   ```
   $pbsserver sham-eth1
   $logevent 127
   $restricted *.ncyu.edu.tw
   $usecp *:/home /home
   $usecp *:/work /work
   $usecp *:/srv /srv
   ```

   The PBS server uses the internal network interface `sham-eth1` to communicate with the clients. The three lines starting with `$usecp` are used because directories `/home`, `/work`, and `/srv` of the server are NFS-mounted on each client. It is more efficient to use the `cp` command to transfer files on these directories between server and client nodes than to use the network interface. Unnecessary network load is avoided.

6. Add the following line to user's `~/.bash_profile` for the man pages and commands to be accessible.

```
setenv PATH ${PATH}:/opt/torque/2.4.5/bin
setenv MANPATH "$MANPATH":/opt/torque/2.4.5/man
```

7. Run PBS for the first time with the following commands to bring up and configure `pbs_server`.

```
/opt/torque/2.4.5/sbin/pbs_server -t create
/opt/torque/2.4.5/sbin/pbs_sched
```

8. Create the queue configuration file `/opt/torque/2.4.5/queue-2.4.5` and configure queue properties by

```
/opt/torque/2.4.5/bin/qmgr < /opt/torque/2.4.5/queue-2.4.5
```

   The contents of `/opt/torque/2.4.5/queue-2.4.5` are listed below.

```
#
# Create queues and set their attributes.
#
#
# Create and define queue default
#
create queue default
set queue default queue_type = execution
set queue default priority = 80
set queue default max_running = 32
set queue default resources_max.cput = 1440:00:00
set queue default resources_min.cput = 100:00:01
set queue default resources_default.cput = 100:00:01
set queue default resources_default.walltime = 240:00:00
set queue default max_user_run = 32
set queue default enabled = true
set queue default started = true
#
# Create and define queue batch
#
create queue batch
set queue batch queue_type = execution
set queue batch priority = 60
set queue batch max_running = 32
set queue batch resources_max.cput = 1440:00:00
set queue batch resources_min.cput = 100:00:01
set queue batch resources_default.cput = 1440:00:01
set queue batch resources_default.walltime = 240:00:00
set queue batch max_user_run = 32
set queue batch enabled = true
set queue batch started = true
```

```
#
# Set server attributes
#
set server scheduling = true
set server acl_host_enable = true
set server acl_hosts = sham
set server managers = root@sham.ncyu.edu.tw
set server managers += quantum@sham.ncyu.edu.tw
set server operators = root@sham.ncyu.edu.tw
set server operators += quantum@sham.ncyu.edu.tw
set server default_queue = default
set server log_events = 127
set server scheduler_iteration = 600
set server node_check_rate = 150
set server tcp_timeout = 6
set server mom_job_sync = true
set server keep_completed = 300
set server submit_hosts = sham
```

In this queue property file, two job queues, `default` and `batch`, are defined. The server attributes are also set at the end. Check and save the queue configuration by the following two commands, respectively.

```
/opt/torque/2.4.5/bin/qmgr -c "list server"
/opt/torque/2.4.5/bin/qmgr -c "print server" > \
             /opt/torque/2.4.5/queue-2.4.5.saved
```

9. Create the `pbs_server` boot script file `/etc/init.d/pbs_server` by the following steps.

```
cd /opt/torque/2.4.5/source/torque-2.4.5/contrib/
cp init.d/suse.pbs_server /etc/init.d/pbs_server.orig
cp /etc/init.d/pbs_server.orig /etc/init.d/pbs_server
chmod 755 /etc/init.d/pbs_server
chmod 644 /etc/init.d/pbs_server.orig
```

Change the following lines in the present `/etc/init.d/pbs_server` file,

```
PBS_DAEMON=/opt/torque/2.4.5/sbin/pbs_server
PBS_HOME=/var/spool/torque
```

Make symbolic links by clicking on YaST → System → System Service (Runlevel) → Enable `pbs_server`. The following links are made by the above administration step.

```
/etc/init.d/rc[2,3,5].d/K01pbs_server -> ../pbs_server
/etc/init.d/rc[2,3,5].d/S07pbs_server -> ../pbs_server
```

10. Create the `pbs_sched` boot script file `/etc/init.d/pbs_sched` by the following steps.

```
cd /opt/torque/2.4.5/source/torque-2.4.5/contrib/
cp init.d/suse.pbs_sched /etc/init.d/pbs_sched.orig
cp /etc/init.d/pbs_sched.orig /etc/init.d/pbs_sched
chmod 755 /etc/init.d/pbs_sched
chmod 644 /etc/init.d/pbs_sched.orig
```

Change the following lines in the current `/etc/init.d/pbs_sched` file,

```
PBS_DAEMON=/opt/torque/2.4.5/sbin/pbs_sched
PBS_HOME=/var/spool/torque
```

Make symbolic links by clicking on YaST → System → System Service (Runlevel) → Enable `pbs_sched`. The following links are established after this administration step.

```
/etc/init.d/rc[2,3,5].d/K01pbs_sched -> ../pbs_sched
/etc/init.d/rc[2,3,5].d/S07pbs_sched -> ../pbs_sched
```

11. Create the server name file `/var/spool/torque/server_name` containing the following lines.

```
sham
#sham-eth1
```

The line starting with `#` is a comment. This step can also be done by command

```
cp /var/spool/torque/server_priv/server_name \
   /var/spool/torque/server_name
```

12. Copy the directory `/var/spool/torque` of server to each node with the following script.

```
for i in 12 13 14
do
  cp -a /var/spool/torque \
        /tftpboot/nodes/192.168.1.$i/var/spool
  cd /tftpboot/nodes/192.168.1.$i/var/spool/torque/
  chmod o+t spool undelivered
done
```

This step is necessary because the `/var` directory of each client node is mounted from the `/tftpboot/nodes/192.168.1.$i/var/` directory of the server in the DRBL construct. Files `/var/spool/torque/server_name` and `/var/spool/torque/mom_priv/config` copied in this step are useful for the client.

13. Start `pbs_mom` daemon on each client manually by the following commands.

```
ssh xeon[12-14]
su -
/opt/torque/2.4.5/sbin/pbs_mom
ps aux | grep pbs_mom
exit
```

In this step, the super user logs on each client from the server, starts and checks the daemon, and exits the client.

14. Create the `pbs_mom` boot script file `/etc/init.d/pbs_mom` with the following steps.

```
cd /opt/torque/2.4.5/source/torque-2.4.5/contrib/
cp init.d/suse.pbs_mom /etc/init.d/pbs_mom.orig
cp /etc/init.d/pbs_mom.orig /etc/init.d/pbs_mom
chmod 755 /etc/init.d/pbs_mom
chmod 644 /etc/init.d/pbs_mom.orig
```

Change the following lines in the present `/etc/init.d/pbs_mom` file.

```
PBS_DAEMON=/opt/torque/2.4.5/sbin/pbs_mom
PBS_HOME=/var/spool/torque
```

Make symbolic links by clicking on YaST → System → System Service (Runlevel) → Enable `pbs_mom`. The following links are created in this steps.

```
/etc/init.d/rc[2,3,5].d/K01pbs_mom -> ../pbs_mom
/etc/init.d/rc[2,3,5].d/S10pbs_mom -> ../pbs_mom
```

15. For the client to start `pbs_mom` at boot time, change the following line in `/opt/drbl/conf/client-extra-service`

```
service_extra_added="ntp pbs_mom"
```

and rerun

```
/opt/drbl/sbin/drblpush -c /etc/drbl/drblpush.conf
```

Alternatively, use the following commands to set up the boot script file on each client,

```
for i in 12 13 14
do
  cp -a /etc/init.d/pbs_mom* \
        /tftpboot/nodes/192.168.1.$i/etc/init.d
  cd /tftpboot/nodes/192.168.1.$i/etc/init.d/rc3.d
  ln -s ../pbs_mom K01pbs_mom
  ln -s ../pbs_mom S10pbs_mom
done
```

The link to `/etc/init.d/pbs_mom` of each client is thus established.

This step is necessary because the `/etc/` directory of each client node is mounted from the `/tftpboot/nodes/192.168.1.$i/etc/` of the server in the full DRBL mode.

16. Reboot the system and check if `pbs_server` and `pbs_sched` are running on the server and if `pbs_mom` is running on the clients by the following commands

```
ps aux | grep pbs
drbl-doit ps aux | grep pbs
```

17. Add the following lines to user's `~/.bash_profile`

```
# Environment for Torque
PATH=$PATH:/opt/torque/2.4.5/bin
export PATH
MANPATH=$MANPATH:/opt/torque/2.4.5/man
export MANPATH
```

for Torque executables and manual pages to be accessible in the search path.

18. Save administrator's guide to `/opt/torque/2.4.5/source/torque-2.4.5/doc/TORQUE_Administrators_Guide-2.4.pdf` for future reference.

These steps complete the installation of the Torque (openPBS) queuing system on this computing cluster.

The scheduler provided by Torque (openPBS), `pbs_sched` is not as versatile as the policy engine, Maui Scheduler[13]. The Torque scheduler `pbs_sched` can optionally be replaced by the Maui Scheduler. The installation steps are presented in the next subsection.

Testing

The queuing system, Torque (openPBS) is tested in this subsection.

Run command

```
/opt/torque/2.4.5/bin/pbsnodes
```

If "state=free" is reported for each client, the system is ready for more complicated tests.

Batch scripts will be used to submit jobs to the queuing system. Three examples on batch scripts will be demonstrated in the following paragraphs.

First, a simple batch example, `test.sh` is created with the following contents.

```
#!/bin/sh
### Job name
#PBS -N TEST
### Output files
#PBS -e test.err
#PBS -o test.log
### Quene name
#PBS -q default
### Number of nodes
#PBS -l nodes=1:ppn=8

# Display the working directory of this job
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

echo Running on host ‘hostname‘
echo Time is ‘date‘
echo Directory is ‘pwd‘

# Run your executable program
/bin/date
#!/bin/sh
### Job name
```

```
#PBS -N TEST
### Output files
#PBS -e test.err
#PBS -o test.log
### Quene name
#PBS -q default
### Number of nodes
#PBS -l nodes=1:ppn=8

# Display the working directory of this job
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

echo Running on host ‘hostname‘
echo Time is ‘date‘
echo Directory is ‘pwd‘

# Run your executable program
/bin/date
```

The lines starting with `#PBS` are options to the job submission command, `qsub`. Run and view the test with the following commands.

```
qsub test.sh
qstat -f
less test.out
less test.err
```

Second, copy `/opt/mpich2/1.2.1/intel-11.1.046/share/examples_ graphics/cpi` and `/etc/mpd.mach` to a work directory, and create a batch file, `mpi.sh` with the following contents.

```
#!/bin/sh
### Job name
#PBS -N MPICH2
### Output files
#PBS -e mpi.err
#PBS -o mpi.log
### Quene name
#PBS -q default
### Number of nodes
#PBS -l nodes=3:ppn=8

echo Starting at ‘hostname‘ on ‘date‘
if [ -n "$PBS_NODEFILE" ]; then
    if [ -f $PBS_NODEFILE ]; then
        # print the node names
        echo "Nodes used for this job:"
        cat ${PBS_NODEFILE}
    fi
fi

# Display the job’s working directory
```

```
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

# Use mpiexec to run MPICH2 programs
mpiexec -machinefile ./mpd.mach -n 24 cpi

# Print end time
echo Job ends at ‘date‘
```

Before submitting this script, the mpich2 ring daemons have to be booted with commands

```
~/bin/mpich2-mpdboot
```

or

```
mpdboot -n 4 -f /etc/mpd.hosts --ncpus=8 --ifhn=sham-eth1
```

The machinefile file in command `mpiexec` has to be copied to the local directory and use relative path. Run and view the test by

```
qsub -q batch mpi.sh
qstat -f
less mpi.out
less mpi.err
```

Third, create and following test batch, `mpich2.sh` with contents below.

```
#!/bin/sh
### Job name
#PBS -N MPICH2
### Output files
#PBS -e mpich2.err
#PBS -o mpich2.log
### Quene name
#PBS -q default
### Number of nodes
#PBS -l nodes=3:ppn=8

echo Starting at ‘hostname‘ on ‘date‘
if [ -n "$PBS_NODEFILE" ]; then
  if [ -f $PBS_NODEFILE ]; then
    # print the node names
    echo "Nodes used for this job:"
    cat ${PBS_NODEFILE}
  fi
fi

# Display the job’s working directory
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

# Use mpiexec to run MPICH2 programs
sort $PBS_NODEFILE | uniq -c | \
     awk '{ printf ("%s:%s\n", $2, $1) ; }' > mpd.mach
mpdboot -f mpd.mach -n `cat $PBS_NODEFILE | uniq | \
     wc -l` --ncpus=8
mpiexec -n `cat $PBS_NODEFILE | wc -l` cpi
mpdallexit

# Print end time
echo Job ends at `date`
```

In this test script, the ring daemon of MPICH2 is started before the execution of the parallelized binary, `cpi`, and terminated after its execution. Hence, it does not need to manually start the ring daemon with command `~/bin/mpich2-mpdboot`. The assignment of the nodes to be used my this script batch is controlled by Torque using the line "`#PBS -l nodes=3:ppn=8`." Unlike the `~/bin/mpich2-mpdboot` command, the option "`--ifhn=sham-eth1`" is not used in the line beginning with `mpdboot` for the following reasons. In the present batch script, the daemon `mpd` is started on the client nodes where `sham-eth1` is not available. In the previous example, `mpd` is started by `~/bin/mpich2-mpdboot` on the server, hence, the network interface of the server has to be specified.

The fact that the CPUs employed by this computing cluster are multi-cored has been taken into account in the above batch script. With this batch script, the cores on the same node will always be assigned to the same task as long as it is possible. This feature is achieved by the `sort` command and the `-f mpd.mach` option of command `mpiexec`.

Run and view this test by

```
qsub -q batch mpich2.sh
qstat -f
less mpich2.out
less mpich2.err
```

3.9 Scheduler: Maui

Maui is a scheduler that can be used to replace the default scheduler of Torque (openPBS), `pbs_sched`. On this computing cluster, Maui Scheduler is installed to the system with the following steps.

1. Unpack `maui-3.2.6p21.tar.gz` to `/tmp/maui-3.2.6p21/` as a super user.

2. Change to directory `/tmp/maui-3.2.6p21/`. Configure, make, and install as root with the following commands

   ```
   cd /tmp/maui-3.2.6p21
   PATH=$PATH:/opt/torque/2.4.5/bin
   export PATH
   ./configure
   make
   make install
   ```

   The Maui Scheduler version 3.2.6p21 package is installed to `/usr/local/maui`. If this step were performed with a regular user, maui could be run by the user with command "`/usr/local/maui/sbin/maui`," but the super user could not run it. Therefore, performing this step as a super user is required.

3. Change that the following lines of the file `/usr/local/maui/maui.cfg`

   ```
   # primary admin must be first in list
   ADMIN1                root quantum
   ```

   The super user, root is the primary administrator of Maui. The regular user, quantum is added as the secondary administrator so that the regular user account can also interact with Torque.

4. Check that the administrators listed in `$MAUIADMIN` are also Torque (openPBS) administrators. This can be confirmed by checking if the file `/opt/torque/2.4.5/queue-2.4.5` applied to `qmgr` contains the following lines.

   ```
   set server managers = root@sham.ncyu.edu.tw
   set server managers += quantum@sham.ncyu.edu.tw
   set server operators = root@sham.ncyu.edu.tw
   set server operators += quantum@sham.ncyu.edu.tw
   ```

5. For a non-root user to run Maui Scheduler on this computing cluster, the files in `/var/spool /torque/mom_priv/config`

   must be world-readable and contain the line

   ```
   $restricted *.ncyu.edu.tw
   ```

   Change their permissions by

   ```
   chmod 755 /var/spool/torque/mom_priv
   chmod 644 /var/spool/torque/mom_priv/config
   ```

6. Check that the `default` queue is defined in `/opt/torque/2.4.5/queue- 2.4.5`. This can be confirmed by checking if the file contains the following line.

   ```
   set server default_queue = default
   ```

7. Check that the following line is in `/opt/torque/2.4.5/queue-2.4.5`

   ```
   set server scheduling = true
   ```

   and the file is applied to `qmgr`. This will allow Maui to utilize Torque scheduling port to obtain real-time event information on job and node transitions.

8. Disable `pbs_sched` at boot time by clicking on YaST → System → System Services (Runlevel) → Select and disable `pbs_sched`. The functionality of the Torque (openPBS) scheduler, `pbs_sched` will be replaced by the Maui Scheduler.

9. Change or check the following lines are in `/usr/local/maui/maui.cfg`.

   ```
   RMCFG[SHAM] TYPE=PBS
   SERVERPORT            42559
   ```

The first line specifies PBS as the resource manager. The second line specifies that Maui uses port `SERVERPORT` for user-scheduler communication. Use the following commands to confirm that PBS and Maui are not using overlapping ports.

```
sudo netstat -pl | grep -i pbs
sudo netstat -pl | grep -i maui
```

10. Create file `/usr/local/maui/maui.ck` and change the permissions of some files by the following commands as a super user

    ```
    touch /usr/local/maui/maui.ck
    chmod 644 /usr/local/maui/maui.ck \
              /usr/local/maui/maui.cfg \
              /usr/local/maui/maui-private.cfg
    ```

11. Add the following lines to user's `~/.bash_profile` for the user to be able to access the executables of Maui Scheduler.

    ```
    PATH=$PATH:/usr/local/maui/bin
    export PATH
    ```

12. Test Maui Scheduler by first changing a line in `/usr/local/maui/maui.cfg` to be

    ```
    SERVERMODE            TEST
    ```

    and then start Maui manually by command

    ```
    /usr/local/maui/sbin/maui
    ```

    Test to see if Maui commands, `showq`, `showbf`, `diagnose`, and `showstats` run fine. Change the line in `/usr/local/maui/maui.cfg` back to be

    ```
    SERVERMODE            NORMAL
    ```

    after the testing.

13. Create Maui boot script by the following commands.

    ```
    su -
    cd /tmp/maui-3.2.6p21/contrib/
    cp /service-scripts/suse.maui.d /etc/init.d/maui.d
    chmod 755 /etc/init.d/maui.d
    ```

    Change the following line in `/etc/init.d/maui.d` to point the correct path

    ```
    MAUI=/usr/local/maui/sbin/maui
    ```

14. Enable automatic starting of `maui.d` on the server at boot time by clicking on YaST → System → System Services (Runlevel) → Select and enable `maui.d` at run levels 2, 3, and 5. After this step, the following links are created

```
/etc/init.d/rc2.d/K01maui.d -> ../maui.d*
/etc/init.d/rc2.d/S01maui.d -> ../maui.d*
/etc/init.d/rc3.d/K01maui.d -> ../maui.d*
/etc/init.d/rc3.d/S01maui.d -> ../maui.d*
/etc/init.d/rc5.d/K01maui.d -> ../maui.d*
/etc/init.d/rc5.d/S01maui.d -> ../maui.d*
```

Reboot the system, and check to see if Maui is running by

```
ps aux | grep -i maui
```

15. Copy the source to /usr/local/maui/source/ after "`make clean`" for further reference. The documentations are in directory `/usr/local/maui/source/maui- 3.2.6p21/docs/`.

These steps complete the installation of the Maui Scheduler. The Torque (openPBS) scheduler is thus replaced by the Maui Scheduler on this computing cluster.

3.10 Monitoring System: Ganglia

For convenient monitoring of the resources of the computing cluster, a distributed monitoring system, Ganglia[14] is installed. Ganglia consists of a monitoring daemon (`gmond`), a meta daemon (`gmetad`), and a PHP web front end. The monitoring daemon `gmond` has to be installed on each client and the server. The meta daemon `gmetad` and the PHP web front only have to be installed on the server. The meta daemon on the server collects informations from the monitoring daemon on each node, and prepares data viewable by the web front end.

The Ganglia package `ganglia-3.1.2.tar.gz` on this computing cluster with the following steps.

1. Several packages are required to be installed on the system before building Ganglia, including `rrdtool-1.3.4-1.27.1`, `rrdtool-devel-1.3.4-1.27.1`, `libconfuse0-2.5-1.63`, `libconfuse-devel-2.5-1.63`, and `python-devel-2.6.0-2.22.1`. These packages are all installed by clicking on YaST → Software → Software Management. After installing the two `rrdtool` packages, check that the file `/usr/include/rrd.h` and the library `/usr/lib64/librrd.so` are on the system. After the installation of the `python-devel` package, check that the file `/usr/include/python2.6/Python.h` is on the system.

2. Unpack package `ganglia-3.1.2.tar.gz` at `/tmp/ganglia-3.1.2/` as a super user.

3. Configure, make, and install the package as follows.

```
cd /tmp/ganglia-3.1.2/
./configure --with-gmetad --with-librrd=/usr/include \
            -prefix=/opt/ganglia/3.1.2
make
make install
```

The package is installed to `/opt/ganglia-3.1.2/`.

4. Copy the manual pages to the proper directories by

```
mkdir /opt/ganglia/3.1.2/man/ \
      /opt/ganglia/3.1.2/man/man1 \
      /opt/ganglia/3.1.2/man/man5
cp /tmp/ganglia-3.1.2/mans/* \
   /opt/ganglia/3.1.2/man/man1/
cp /tmp/ganglia-3.1.2/gmond/gmond.conf.5 \
   /opt/ganglia/3.1.2/man/man5
```

Add the following lines to user's `~/.bash_profile` for the executables and manual pages of the Ganglia to be accessible by the user.

```
# Environment for ganglia
PATH=$PATH:/opt/ganglia/3.1.2/bin
export PATH
MANPATH=$MANPATH:/opt/ganglia/3.1.2/man
export MANPATH
```

5. Establish the boot script for `gmond` as a super user by the following commands.

```
cd /tmp/ganglia/3.1.2/
cp ./gmond/gmond.init.SuSE /etc/init.d/gmond
cp /etc/init.d/gmond /etc/init.d/gmond.orig
chmod 755 /etc/init.d/gmond
chmod 644 /etc/init.d/gmond.orig
```

Change a line in `/etc/init.d/gmond` to point to the right path for the executable

```
GMOND_BIN=/opt/ganglia/3.1.2/sbin/gmond
```

and comment out a few lines as follows.

```
# Determine the base and follow a runlevel link name.
#base=${0##*/}
#link=${base#*[SK][0-9][0-9]}

# Force execution if not called by a runlevel directory.
#test $link = $base && START_GMOND=yes
#test "$START_GMOND" = yes || exit 0
```

because these lines cause some troubles on the clients, and have to be excluded. Create the boot script links by

```
/sbin/chkconfig --add gmond
```

Check the links with command

```
/sbin/chkconfig --list gmond
```

6. On the server node, configure `gmond` as a super user with the following commands.

```
mkdir /etc/ganglia
cd /etc/ganglia
/opt/ganglia/3.1.2/sbin/gmond --default_config \
                           > ./gmond.conf.orig
cp gmond.conf.orig gmond.conf
```

Change the following lines in `/etc/ganglia/gmond.conf`,

```
cluster {
  name = "xeoncluster_sham"
  owner = "Computational Physics Laboratories"
  latlong = "N23.28 E120.29"
  url = "http://sham.ncyu.edu.tw/"
}
udp_send_channel {
  mcast_join = 192.168.1.15
  port = 8649
  ttl = 1
}
udp_recv_channel {
/* mcast_join = 192.168.1.15 */
port = 8649
/* bind = 192.168.1.15 */
}
```

7. Copy directory `/etc/ganglia/` of server to each client with the following script.

   ```
   for i in 12 13 14
   do
     cp -a /etc/ganglia /tftpboot/nodes/192.168.1.$i/etc/
     chmod 755 /tftpboot/nodes/192.168.1.$i/etc/ganglia
     chmod 644 /tftpboot/nodes/192.168.1.$i/etc/ganglia/*
   done
   ```

   This step is necessary because the `/etc` directory of each client node is mounted from the `/tftpboot/nodes/192.168.0.$i/etc/` of the server. Files `/etc/ganglia/gmond.conf*` are copied to each node in the process.

8. Service `/etc/init.d/gmond` on each client is configured to start automatically at boot time by the following method. Change the following line in `/opt/drbl/conf/client-extra-service`

   ```
   service_extra_added="ntp pbs_mom gmond"
   ```

   and rerun

   ```
   /opt/drbl/sbin/drblpush -c /etc/drbl/drblpush.conf
   ```

   Alternatively, this step can be achieved by the following script to set up the initialization script of `gmond` on each client.

```
for i in 12 13 14
do
  cp -a /etc/init.d/gmond* \
        /tftpboot/nodes/192.168.1.$i/etc/init.d
  cd /tftpboot/nodes/192.168.1.$i/etc/init.d/rc3.d
  ln -s ../gmond S99gmond
done
```

This step is necessary because the `/etc` directory of each client node is mounted from the directory `/tftpboot/nodes/192.168.1.$i/etc/` of the server. Check that `gmond` is up on the clients by

```
drbl-doit ps aux | grep gmond
```

after reboot.

9. Establish the boot script for `gmetad` on the server as a super user with the following commands.

   ```
   cp ./gmetad/gmetad.init.SuSE /etc/init.d/gmetad
   cp /etc/init.d/gmetad /etc/init.d/gmetad.orig
   chmod 755 /etc/init.d/gmetad
   chmod 644 /etc/init.d/gmetad.orig
   ```

   Change a line in file `/etc/init.d/gmetad` to point to the right path for the executable.

   ```
   GMETAD_BIN=/opt/ganglia/3.1.2/sbin/gmetad
   ```

   Create the boot script links by

   ```
   /sbin/chkconfig --add gmetad
   ```

   Check the links with command

   ```
   /sbin/chkconfig --list gmetad
   ```

10. Configure the initialization script `/etc/init.d/gmetad` with commands.

    ```
    cp /tmp/ganglia-3.1.2/gmetad/gmetad.conf /etc/ganglia/
    chmod 644 /etc/ganglia/gmetad.conf
    cp -a /etc/ganglia/gmetad.conf \
          /etc/ganglia/gmetad.conf.orig
    ```

    Change a line in the configuration file `/etc/ganglia/gmetad.conf`.

    ```
    data_source "xeoncluster_sham" localhost
    ```

    Create the round-robin database of `gmetad` by

    ```
    cd /var/lib
    mkdir -p ganglia/rrds
    chown nobody.nobody /var/lib/ganglia/rrds
    ```

The round-robin database of service `gmetad` is located at the directory `/var/lib/ganglia/rrds`.

11. Check that services `gmond` and `gmetad` are properly configured to start automatically at boot time by clicking on YaST2 → System → System Services (Runlevel).

12. Install the PHP web front for Ganglia with the following commands.

```
mkdir /srv/www/htdocs/ganglia
cp -a /tmp/ganglia/ganglia-3.1.2/web/* \
      /srv/www/htdocs/ganglia
chown -R root.root /srv/www/htdocs/ganglia/*
chmod 644 /srv/www/htdocs/ganglia/conf.php
chmod 644 /srv/www/htdocs/ganglia/version.php
```

    Directory `/srv/www/htdocs/` is used because of the following line

```
 DocumentRoot "/srv/www/htdocs"
```

    in the Apache configuration file `/etc/apache2/default-server.conf` of this cluster.

13. Activate the HTTP server by clicking on YaST → Network Services → HTTP Server. Set "Listen on Ports 140.130.91.204/80, 192.168.1.15/80, and 127.0.0.1/80." Select "Enable PHP5 Scripting." Set Main Host as follows: "Server Name/sham" and "Server Administrator Email/root@sham."

14. Enable apache2 by clicking on YaST → System → System Services (Runlevel).

15. Test Ganglia by visiting the following URL with the command.

```
firefox http://localhost/ganglia/
```

16. Copy the Ganglia source to `/opt/ganglia/source/ganglia-3.1.2/` for future reference after "`make clean`".

This completes the installation of the monitoring system, Ganglia on this computing cluster.

Chapter 4

The Compeleted Cluster

The completed computing cluster in full function is shown in Fig. 4.1. The cluster has one server node and three clients connected by gigabit network. The operating system is based on openSUSE Linux 11.1 distribution. Both Intel and Open64 compilers are installed. DRBL is used to set up the networking system between nodes. In the process, DHCP, TFTP, NFS, and NIS are installed, if not previously, and configured automatically. NTP and SSH are used for the nodes to work in unison. Torque (openPBS) and Maui Scheduler are employed for the queuing system. The system resources are monitored by Ganglia monitoring system. An example of the web display of system resource usage monitored by Ganglia is given in Fig. 4.2. The parallelization is achieved by MPICH2 or MPICH. The performance testing codes of MPICH show that the performances of MPICH compiled by Intel and Open64 compilers are about the same.

This computing cluster has diskless clients; all system and user files are on the server storage disks. This implementation simplifies the system installation and maintenance, and improves hardware reliability. However, since all system and user files are on the server only, important files must be backed up regularly to avoid loss of data caused by the failure of server hard disks. Notice that there is a partition, `/srv` that is physically located on device `/dev/sdc`. This partition is intentionally allocated to a hard disk that is different from the the device `/dev/sda` accommodating `/`, `/home`, `/opt`, and other directories. Hence, important files on device `/dev/sda` can be regularly copied to device `/dev/sdc`. Unless these two disks fail at the same time, the loss of data due to server hard disk failure is reduced to minimum.

The construction of the high-performance computing cluster is thus completed, and the

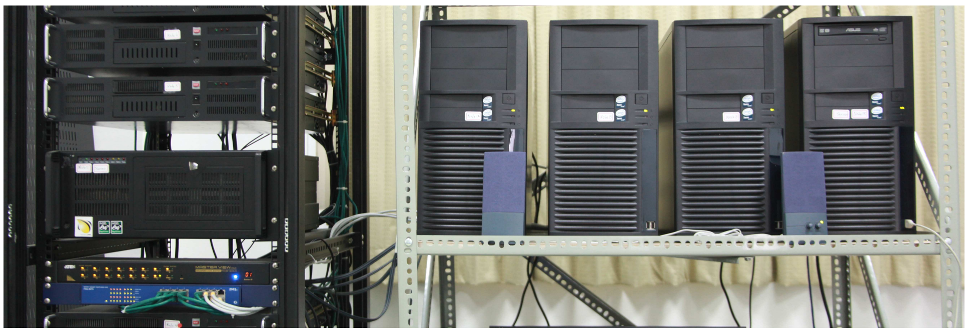
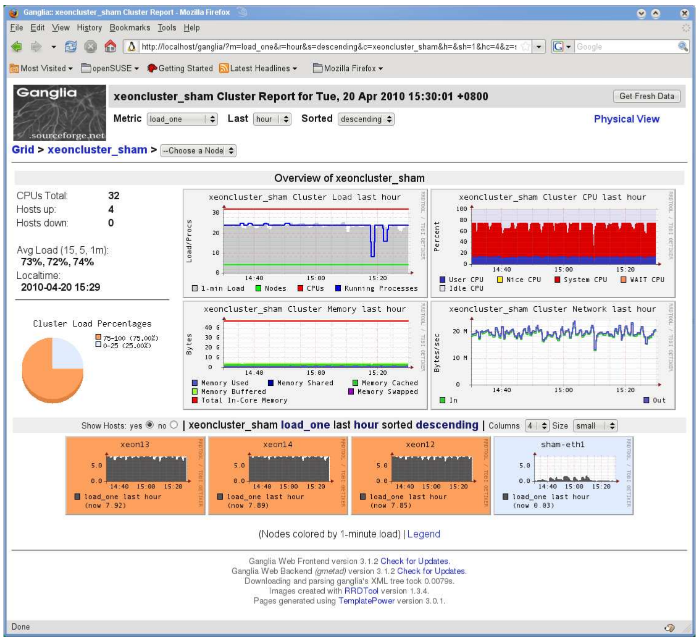


Figure 4.1: The four computers at the right hand side of the picture are the server and the three clients of the high-performance computing cluster. The gigabit switch and the KVM controller are at the lower left corner of the picture. Currently, this Xeon cluster is sharing the gigabit switch and the KVM controller with a Opteron cluster. All other hardware in the picture belongs to the Opteron cluster.


Ganglia:: xeoncluster_sham Cluster Report - Mozilla Firefox

http://localhost/ganglia/?m=load_one&r=hour&s=descending&c=xeoncluster_sham&h=&sh=1&hc=4&z=

xeoncluster_sham Cluster Report for Tue, 20 Apr 2010 15:30:01 +0800

Metric load_one Last hour Sorted descending

Grid > xeoncluster_sham > --Choose a Node

Overview of xeoncluster_sham

CPUs Total: 32
Hosts up: 4
Hosts down: 0

Avg Load (15, 5, 1m):
73%, 72%, 74%
Localtime:
2010-04-20 15:29

Show Hosts: yes [✓ ●] no ○ | xeoncluster_sham **load_one** last **hour** sorted **descending** | Columns 4 Size small

xeon13: load_one last hour (now 7.92)
xeon14: load_one last hour (now 7.89)
xeon12: load_one last hour (now 7.85)
sham-eth1: load_one last hour (now 0.03)

(Nodes colored by 1-minute load) | Legend

Ganglia Web Frontend version 3.1.2 Check for Updates.
Ganglia Web Backend *(gmetad)* version 3.1.2 Check for Updates.
Downloading and parsing ganglia's XML tree took 0.0079s.
Images created with RRDTool version 1.3.4.
Pages generated using TemplatePower version 3.0.1.

Done

File Edit View History Bookmarks Tools Help
Most Visited openSUSE Getting Started Latest Headlines Mozilla Firefox
Ganglia .sourceforge.net
Get Fresh Data
Physical View
xeoncluster_sham Cluster Load last hour
Load/Procs
1-min Load Nodes CPUs Running Processes
xeoncluster_sham Cluster CPU last hour
Percent
User CPU Nice CPU System CPU WAIT CPU Idle CPU
Cluster Load Percentages
75-100 (75.00%)
0-25 (25.00%)
xeoncluster_sham Cluster Memory last hour
Bytes
Memory Used Memory Shared Memory Cached Memory Buffered Memory Swapped Total In-Core Memory
xeoncluster_sham Cluster Network last hour
Bytes/sec
In Out
14:40 15:00 15:20

Figure 4.2: The web display of system resource usage monitored by Ganglia.

computing cluster is ready for the installation of parallelized applications.

Chapter 5

Application Software

The parallelize application, Vienna *Ab-initio* Simulation Package (VASP) is installed on this computing cluster for nanoscience simulation. VASP is a package for performing first-principle molecular simulations using pseudopotentials or the projector-augmented wave method and a plane wave basis set[27]. Notice that VASP is not public-domain or share-ware, a fee has to be paid to license the software[28].

Before the installation of VASP, the FFT package `fftw-3.2.2.tar.gz` is installed with the following steps[29]. This FFT package will be called by VASP.

1. Unpack `fftw-3.2.2.tar.gz` as super user at a work directory.

   ```
   tar -xvzpf fftw-3.2.2.tar.gz
   ```

2. Configure, build, and install the library by commands

   ```
   ./configure --prefix=/opt/fftw/3.2.2 CC=icc F77=ifort
   make
   make install
   ```

   The compiled library is then installed to `/opt/fftw/3.2.2/`.

3. Copy the source to `/opt/fftw/3.2.2/source/` for future reference. The documentation in HTML format starts at `/opt/fftw/3.2.2/source/doc /html/index.html`.

VASP version 4.6.36 is compiled by Intel compilers version 11.1.046 and is installed on this computing cluster using sources `vasp.4.6.tar.gz` and `vasp.4.lib.tar.gz`. Four different kinds of VASP binary codes are built on this cluster: parallel, serial, parallel gamma-point-only, and serial gamma-point-only VASP[30]. The steps to build these VASP binary codes are given in the following subsections.

5.1 Parallel VASP

The steps used to build the parallelized VASP binary code are presented below.

1. Build the library in `~/SIMUcode/VASP/4.6.36/src/vasp.4.lib` as follows.

   Select `makefile.linux_ifc_P4` as the starting `Makefile` by

   ```
   cd ~/SIMUcode/VASP/4.6.36/src/vasp.4.lib
   cp makefile.linux_ifc_P4 Makefile.orig
   cp -a Makefile.orig Makefile
   ```

Change a line in `Makefile`

```
FC=ifort
```

Issue command

```
make
```

The `vasp.4.lib` is thus built after this step.

2. Build the parallelized executable code for VASP as follows.

   Select `makefile.linux_ifc_P4` as the starting `Makefile` by

```
cd ~/SIMUcode/VASP/4.6.36/src/vasp.4.6
cp makefile.linux_ifc_P4 Makefile.orig
cp -a Makefile.orig Makefile
```

   A few important lines in `Makefile` have to be changed as follows.

```
FC=ifort
#CPP     = $(CPP_)  -DHOST=\"LinuxIFC\" \
#          -Dkind8 -DNGXhalf -DCACHE_SIZE=12000
#          -DPGF90 -Davoidalloc
#OFLAG=-O3 -xW -tpp7
OFLAG=-O3 -xSSE4.1
#BLAS=  /opt/libs/libgoto/libgoto_p4_512-r0.6.so
BLAS=-L/opt/intel/Compiler/11.1/046/mkl/lib/em64t \
     -lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
     -lpthread
#LAPACK= ../vasp.4.lib/lapack_double.o
LAPACK=-L/opt/intel/Compiler/11.1/046/mkl/lib/em64t \
       -lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
       -lpthread
#FFT3D   = fftw3d.o fft3dlib.o \
#          /opt/libs/fftw-3.0.1/lib/libfftw3.a
#FFT3D   = fftw3d.o fft3dlib.o \
#          /opt/fftw/3.2.2/lib/libfftw3.a
FC=mpif90
FCL=$(FC)
CPP    = $(CPP_) -DMPI  -DHOST=\"LinuxIFC\" -DIFC \
        -Dkind8 -DNGZhalf -DCACHE_SIZE=16000 \
        -DPGF90 -Davoidalloc -DMPI_BLOCK=500
#FFT3D   = fftmpiw.o fftmpi_map.o fft3dlib.o  \
#         /opt/intel/Compiler/11.1/046/mkl/interfaces/
#         fftw3xf/libfftw3xf_intel.a
#fft3dlib.o : fft3dlib.F
#     $(CPP)
#     $(FC) -FR -lowercase -O1 -tpp7 -xW -unroll0 \
#           -e95 -vec_report3 -c $*$(SUFFIX)
FFT3D   = fftmpiw.o fftmpi_map.o fft3dlib.o \
          /opt/fftw/3.2.2/lib/libfftw3.a
$(FC) -FR -lowercase -O1 -xSSE4.1 -unroll0 \
      -warn nostderrors -vec_report3 -c $*$(SUFFIX)
```

Optimization option -xSSE4.1 is used because Xeon CPUs are used in this cluster. BLAS and LAPACK libraries of Intel MKL are used; whereas the FFT3D library is compiled on the server. MPICH2 is used for parallelization. Issue the following command to build the binary code.

```
make
```

Parallelized binary file ~/SIMUcode/VASP/4.6.36/src/vasp.4.6/vasp is generated after this step. Change the executable VASP file vasp to a different name, say vasp-4.6.36_parallel.

5.2 Serial VASP

The steps used to build the parallelized VASP binary code are presented below.

1. Build the library in ~/SIMUcode/VASP/4.6.36/src/vasp.4.lib with the same method as in Subsection 5.1
2. Build the serial executable code for VASP as follows.

   Select makefile.linux_ifc_P4 as the starting Makefile by

   ```
   cd ~/SIMUcode/VASP/4.6.36/src/vasp.4.6
   cp makefile.linux_ifc_P4 Makefile.orig
   cp -a Makefile.orig Makefile
   ```

   A few important lines in Makefile have to be changed as follows.

   ```
   FC=ifort
   CPP     = $(CPP_)  -DHOST=\"LinuxIFC\" \
             -Dkind8 -DNGXhalf -DCACHE_SIZE=16000
             -DPGF90 -Davoidalloc
   #OFLAG=-O3 -xW -tpp7
   OFLAG=-O3 -xSSE4.1
   #BLAS=  /opt/libs/libgoto/libgoto_p4_512-r0.6.so
   BLAS=-L/opt/intel/Compiler/11.1/046/mkl/lib/em64t \
        -lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
        -lpthread
   #LAPACK= ../vasp.4.lib/lapack_double.o
   LAPACK=-L/opt/intel/Compiler/11.1/046/mkl/lib/em64t \
          -lmkl_intel_lp64 -lmkl_sequential -lmkl_core \
          -lpthread
   #FFT3D   = fftw3d.o fft3dlib.o \
   #          /opt/libs/fftw-3.0.1/lib/libfftw3.a
   FFT3D   = fftw3d.o fft3dlib.o \
             /opt/fftw/3.2.2/lib/libfftw3.a
   #FC=mpif90
   FCL=$(FC)
   CPP    = $(CPP_) -DMPI  -DHOST=\"LinuxIFC\" -DIFC \
           -Dkind8 -DNGZhalf -DCACHE_SIZE=16000 \
           -DPGF90 -Davoidalloc -DMPI_BLOCK=500
   #FFT3D   = fftmpiw.o fftmpi_map.o fft3dlib.o  \
   ```

```
#        /opt/intel/Compiler/11.1/046/mkl/interfaces/
#        fftw3xf/libfftw3xf_intel.a
#fft3dlib.o : fft3dlib.F
#     $(CPP)
#     $(FC) -FR -lowercase -O1 -tpp7 -xW -unroll0 \
#           -e95 -vec_report3 -c $*$(SUFFIX)
$(FC) -FR -lowercase -O1 -xSSE4.1 -unroll0 \
      -warn nostderrors -vec_report3 -c $*$(SUFFIX)
```

Optimization option `-xSSE4.1` is used for the Xeon CPUs used with this cluster. Issue the following command to build the binary code.

```
make
```

Serial executable file `~/SIMUcode/VASP/4.6.36/src/vasp.4.6/vasp` is generated after this step. Change the executable VASP file `vasp` to a different name, say `vasp-4.6.36_serial`.

5.3 Parallel Gamma-Point-Only VASP

The steps used to build the parallelized gamma-point-only VASP binary code are presented below.

1. Build the library in `~/SIMUcode/VASP/4.6.36/src/vasp.4.lib` with the same method as in Subsection 5.1

2. Reuse the `Makefile` of the parallel VASP in Subsection 5.1. Change a line in the `Makefile`.

```
CPP    = $(CPP_) -DMPI  -DHOST=\"LinuxIFC\" -DIFC \
         -Dkind8 -DNGZhalf -DwNGZhalf \
         -DCACHE_SIZE=16000 -DPGF90 \
         -Davoidalloc -DMPI_BLOCK=500
```

   Issue command

```
make
```

Parallel gamma-point-only binary VASP code is generated after this step. Rename the executable file `vasp` to be `vasp-4.6.36_gamma_parallel`.

5.4 Serial Gamma-Point-Only VASP

The steps used to build the serial gamma-point-only VASP binary code are presented below.

1. Build the library in `~/SIMUcode/VASP/4.6.36/src/vasp.4.lib` with the same method as in Subsection 5.1

2. Reuse the `Makefile` of the parallel VASP in Subsection 5.2. Change a line in the `Makefile`.

```
    CPP     = $(CPP_)  -DHOST=\"LinuxIFC\" \
            -Dkind8 -DNGXhalf -DwNGXhalf \
            -DCACHE_SIZE=16000 -DPGF90 -Davoidalloc
```

Issue command

```
    make
```

Serial gamma-point-only binary VASP code is generated after this step. Rename the executable file `vasp` to be `vasp-4.6.36_gamma_serial`.

5.5 Running VASP

For convenience of usage, the four kinds of VASP binary codes are copied to directory `/opt/VASP/4.6.36/bin/`. The parallel VASP is made to be the default binary code by

```
ln -s ~/bin/vasp /opt/VASP/4.6.36/bin/vasp-4.6.36_parallel
```

Script file `govasp.sh` with the following contents are created.

```
#!/bin/sh
### Job name
#PBS -N VASP
### Output files
#PBS -e vasp.err
#PBS -o vasp.log
### Quene name
#PBS -q default
### Number of nodes
#PBS -l nodes=3:ppn=8

echo Starting at ‘hostname‘ on ‘date‘
if [ -n "$PBS_NODEFILE" ]; then
  if [ -f $PBS_NODEFILE ]; then
    # print the node names
    echo "Nodes used for this job:"
    cat ${PBS_NODEFILE}
  fi
fi

# Display the job’s working directory
echo Working directory is $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

# Use mpiexec to run MPICH2 programs
sort $PBS_NODEFILE | uniq -c | \
     awk ’{ printf ("%s:%s\n", $2, $1) ; }’ \
     > mpd.mach
mpdboot -f mpd.mach -n ‘cat $PBS_NODEFILE | \
     uniq | wc -l‘ --ncpus=8
mpiexec -n ‘cat $PBS_NODEFILE | wc -l‘ vasp
```

```
mpdallexit

# Print end time
echo Job ends at ‘date‘
```

After copying it to the work directory, the script can be submitted to the queuing system by

```
qsub govasp.sh
```

5.6 VASP benchmark

The example in `bench.Hg.tar.gz` is computed with 1 to 24 CPU cores. The benchmark data obtained with this computing cluster are tabulated in Table 5.1 under the system name of "Dual Quad-Core Xeon Cluster." The benchmark data of other main-stream systems are given in the latter part of the table for comparison.

Table 5.1: VASP benchmark (bench.Hg)

System Name	Number of CPU Cores	CPU Time (sec)
Dual Quad-Core Xeon	1	93.70
Cluster	2	51.95
	4	30.53
	8 (1 node)	24.19
	16 (2 nodes)	23.97
	24 (3 nodes)	230.47
Itanium2 1300 HP-UX	1	127[†]
IBM SP3	1	356[†]
SGI Origin	1	1200[†]
SGI Origin	4	330[†]

[†] Data are cited from Ref. [30]. The middle column is the number of nodes.

On this computing cluster, running the benchmark job with four CPU cores is about three times as fast as running it with a single core. Running the job with 8 cores does not gain much in performance. Running the job with 16 cores is just about as fast as running it with 8 cores. Running the job with 24 cores greatly degrade the performance. The performance saturation at 16 cores and degradation at 24 cores are because communication through the gigabit network slow down the computation. This adverse effect becomes severe as more than two nodes are involved with a computation.

Comparison with the main-stream systems shows that the performance of this computing cluster, to some extent, is comparable with the main-stream systems. The Xeon cluster does not scale as well as SGI Origin when multi-nodes are used for a computation. This can be attributed to the commodity networking hardware used with the Xeon cluster.

Chapter 6

Concluding Remarks

In this monograph, the construction procedures for a high-performance computing cluster with diskless clients are presented. Its applications to nanoscience simulations are also demonstrated.

Commodity hardware and open source software are used for the construction of the computing cluster. The cluster consists of one server node and three clients. All nodes are connected by gigabit network. The cluster operating system is based on openSUSE Linux 11.1 distribution. Both Intel and Open64 compilers are installed.

DRBL is used to set up the network connection between nodes. With DRBL, services DHCP, TFTP, NFS, and NIS are installed and configured automatically. NTP is employed to ensure that the system time is consistent between nodes. SSH are used for node-to-node communication.

The queuing system are implemented using Torque (openPBS) batch system and Maui scheduler.

With the Ganglia monitoring system, the usage of system resources can be monitored using web browser such as `firefox`.

The parallelization is achieved by MPICH2 or MPICH. Comparison of the performance testing codes of MPICH compiled by Intel and Open64 compilers shows that the performance of MPICH is rather independent of the compiler.

Parallelized application, VASP is compiled and run on this computing cluster. Benchmark data show that the performance of this computing cluster is comparable, to some extent, with the main-stream machines such as Itanium2 1300, IBM SP3, and SGI Origin.

This high-performance computing cluster features several advantages, including

- **Easy installation**: With DRBL, the installation process of a high-performance computing cluster is greatly simplified. The network services DHCP, TFTP, NFS, and NIS are installed, if not previously, and configured automatically.

- **Easy maintenence**: All system and application software are installed only on the server hard disk. The clients have no hard disk. Hence, the cluster appears to the administrator to be a single machine; software update and account management are all performed on the server only. The maintenence efforts are therefore reduced once the cluster is properly constructed and configured.

- **Improved hardware reliability**: Because this high-performance computing cluster has diskless clients, the possible problems of hard disk failure are completely avoided on the clients. The hardware reliability of the computing cluster is thus improved.

- **High performance-cost ratio**: Since the computing cluster is constructed with commodity hardware and open source software, its cost is reduced to a minimum. The benchmark data show that the performance of this computing cluster is comparable, in some cases, with main-stream systems. Hence, the performance-cost ratio is quite high.

Because of the above features, the construction methods demonstrated in this monograph are particularly suitable for laboratories that need the computing power of a small- to medium-sized high-performance computing cluster.

Bibliography

[1] R. M. Martin, *Electronic structures: Basic theory and practical methods.* Cambridge: Cambridge University Press, 2005.

[2] P. Ordejon, D. A. Drabold, R. M. Martin, and M. P. Grumbach, “Linear system-size scaling methods for electronic-structure calculations,” *Phys. Rev. B*, vol. 51, no. 3, pp. 1456–1476, 1995.

[3] P. Kitterrick (Apr. 2010), “Building a Diskless Linux Cluster: Debian(Etch) + DRBL + GridEngine.” [Online]. Available: http://blog.padraigkitterick.com/2007/ 07/25/building-a-diskless-linux-cluster-debian-etch-drbl-gridengine/.

[4] For example, Single Linux Image Management(SLIM) is also a solution to high-performance computing clusters with diskless clients. Information is a available at http://slim.csis.hku.hk/.

[5] Intel Corporation (Apr. 2010), “Intel Xeon Processor 5000 Sequence.” [Online]. Available: http://www.intel.com/p/en_US/products/server/processor/xeon5000/.

[6] MiTAC International Corporation (Apr. 2010), “Tyan System Boards Tempest i5100W S5376 (S5376G2NR).” [Online]. Available: http://www.tyan.com/product_SKU_spec.aspx?ProductType=MB&pid=605&SKU=600000019.

[7] openSUSE (Apr. 2010), “openSUSE 11.1.” [Online]. Available: http://www.opensuse.org/.

[8] The GCC team (Apr. 2010), “GCC, the GNU Compiler Collection.” [Online]. Available: http://gcc.gnu.org/.

[9] Intel Software Network (Apr. 2010), “Intel Compilers.” [Online]. Available: http://software.intel.com/en-us/intel-compilers/.

[10] AMD Developer Central (Apr. 2010), “x86 Open64 Compiler Suite.” [Online]. Available: http://developer.amd.com/cpu/open64/Pages/default.aspx.

[11] S. Shiau (Apr. 2010), “DRBL (Diskless Remote Boot in Linux).” [Online]. Available: http://drbl.sourceforge.net/ or http://drbl.nchc.org.tw/.

[12] Cluster Resources Inc. (Apr. 2010), “TORQUE Resource Manager.” [Online]. Available: http://www.clusterresources.com/products/torque-resource-manager.php.

[13] Cluster Resources Inc. (Apr. 2010), “Maui Cluster Scheduler.” [Online]. Available: http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php.

[14] Ganglia (Apr. 2010), “Ganglia Monitoring System.” [Online]. Available: http://ganglia.info/.

[15] Argonne National Laboratory (Apr. 2010), “MPICH2.” [Online]. Available: http://www.mcs.anl.gov/research/projects/mpich2/.

[16] MPICH (Apr. 2010), “MPICH-A Portable Implementation of MPI.” [Online]. Available: http://www.mcs.anl.gov/research/projects/mpi/mpich1/index.htm.

[17] K. Thomas, *Beginning SUSE Linux: From Novice to Professional.* Berkeley: Apress, 2005.

[18] AMD (Apr. 2010), “Graphics Drivers & Software.” [Online]. Available: http://support.amd.com/us/gpudownload/Pages/index.aspx.

[19] C. T. Yang, P. I. Chen, and Y. L. Chen, “Performance evaluation of SLIM and DRBL diskless PC clusters on Fedora Core 3,” *Sixth International Conference on Parallel and Distributed Computing, Applications, and Technologies, 2005(PDCAT 2005)*, pp. 479–482, 2005.

[20] C. T. Yang and Y. C. Chang, “An introduction to a PC cluster with diskless slave nodes,” *Tunghai Science*, vol. 4, pp. 25–46, 2002.

[21] M. H. Chen and T. L. Li, “Construction of a high-performance computing cluster: A curriculum for engineering and science students,” *Comput. Appl. Eng. Educ.*, 2009. Published on-line. DOI: 10.1002/cae.20352.

[22] DHCP (Apr. 2010), “Resources for DHCP.” [Online]. Available: http://www.dhcp. org/.

[23] PXELINUX (Apr. 2010), “PXELINUX.” [Online]. Available: http://syslinux.zytor.com/wiki/index.php/PXELINUX.

[24] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI.* Cambridge: MIT Press, 1999.

[25] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2.* Cambridge: MIT Press, 1999.

[26] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach using BSP and MPI.* Oxford: Oxford University Press, 2004.

[27] G. Kresse and J. Furthmuller, “Efficient iterative schemes for *ab initio* total-energy calculations using plane-wave basis set,” *Phys. Rev. B*, vol. 54, no. 16, pp. 11169–11186, 1996.

[28] VASP Group (Apr. 2010), “Vienna Ab-initio Simulation Package.” [Online]. Available: http://cms.mpi.univie.ac.at/vasp/.

[29] M. Frigo and S. G. Johnson (Apr. 2010), “FFTW.” [Online]. Available: http://www.fftw.org/.

[30] G. Kresse, M. Marsman, and J. Furthmuller, *VASP the Guide.* Vienna, April 23, 2009.