A Debian/Linux Based Beowulf Cluster

Min Xu, Shiliang Hu, Marong Phadoongsidhi, Chris Colletti

December 5, 2000

Abstract

This document serves as the Beowulf bake-off assignment write-up for ECE757 Fall 2000 of University of Wisconsin-Madison. And I also want it to be a brief and complete "cookbook" for building a Beowulf Cluster based on Debian GNU/Linux system.

Contents

1 System Infrastructure **1**

2 System Installation **2**
2.1 General installation 2
2.2 Kernel preparation 3

3 System Maintenance **4**
3.1 NIS/YP and Autofs 4
3.2 rdate and NTP 5
3.3 ruptime 5
3.4 reboot_all 6
3.5 software document 6
3.6 microcode update 6
3.7 add shadow diskless node 6

4 System Security **6**

5 Parallel Computing Environment on The Beowulf Cluster **7**
5.1 Communication Bandwidth Tests 7
5.2 Other Performance Tests 10

1 System Infrastructure

A Beowulf system is more or less a collection of UNIX machines that tightly connected with TCP/IP. From a parallel programmer perspective, the system appears to be a single big system. However, from a system administrator perspective, the system is a special group of UNIX hosts. In Figure 1, our SMP Intel server boxes are connected with standard Ethernet interconnect. Only one of them has local hard disk installed. And this system (Beowulf-1) is serving as the NFS server[1] and holding all the root directories of those diskless nodes. Beowulf-2 is the "Master Diskless Node" with a RW root directory mounted off the NFS server. All other diskless nodes are the shadow copies of the "Master Diskless Node".

This infrastructure allows us to have very different software/hardware configuration between NFS server and those diskless nodes. Generally, the NFS server has fast disks, fast network, multiple moderate CPUs and fully loaded server-wares to server large number of diskless clients, while the diskless clients have high-end CPU

[1] In our implementation, Beowulf-1 is also participating MPICH computation as one of the cluster node. Due to the scale of our cluster, it does not introduce too much load imbalance.

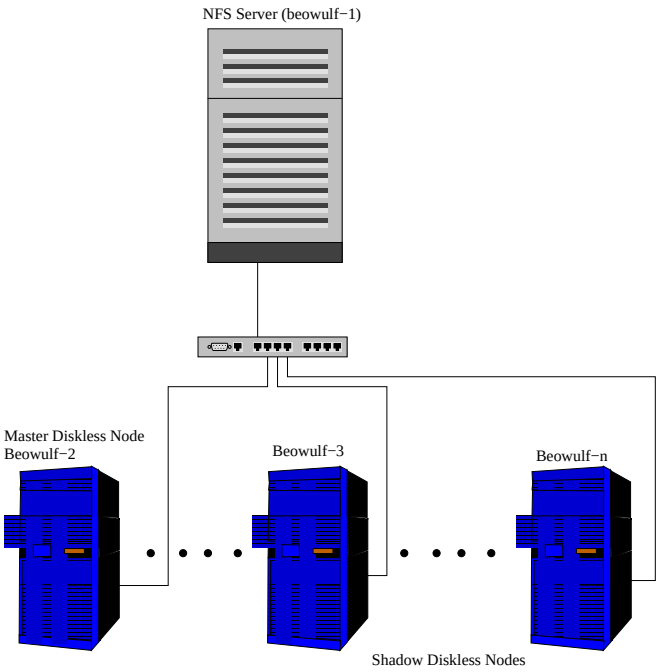

Figure 1: Infrastructure of the Beowulf system. Only Beowulf-1 has local disk installed and it serves as a NFS server for hosting all the file systems of the diskless nodes. Beowulf-2 is the master diskless node, which running a live system mirror of all the diskless node. All other system are the shadow copies of the master diskless node.

and almost no server-wares running on them. As a consequence of this infrastructure, two systems need to be maintained separately. Fortunately, the "diskless" package from Debian took care of the most installation and maintenance tasks of the master and shadow diskless nodes.

2 System Installation

2.1 General installation

Installing the NFS server is straightforward. Just download 5 floppies from *www.Debian.org* and install them to your Internet ready server box. Following the instruction and partition the disk. Note that the /var partition need to be considerably bigger than normal, because it will be holding the diskless file system data. The master node will roughly cost 500MB to 1GB and every shadow system will cost about 5MB to 10MB. When installing the system, you may want to choose the closest server for downloading the installation files. The nearest mirror is ftp.cs.wisc.edu.

After base system is installed, use "tasksel" to coarsely select software packages that you want to install. Obviously, the "parallel-computing" are needed. For a fine selection of software packages, you can use "dselect" to choose individual package from almost 4000 available.

The NFS user space server is required for diskless nodes. Please do:

```
apt-get install nfs-server
```

and if you want, you may remove the default installation of kernel space NFS server by:

```
apt-get remove nfs-kernel-server
```

Now we can prepare the root file system of the master diskless node. First use:

```
apt-get install diskless
```

to install the Debian diskless package. And go to the nearest Debian to grab the base system tarball named base2_2.tgz and simple diskless-image package[2]. Issue the following command:

```
diskless-newimage
```

to create a new master diskless image in /var/lib/diskless/default/root directory. By answering some questions, you will have the master diskless image ready. Then modify /etc/exports to export that directory as RW only to the client from IP of the master node. A example exports file can be found in Beowulf-1:/etc/exports.

To make the booting and kernel upgrade easier, all diskless nodes are boot with "etherboot". To install that, the NFS server needs the "bootp" server and "netboot". To do that:

```
apt-get install bootp netboot
```

And edit the file /etc/bootptab as the example in Beowulf-1:/etc/bootptab. To make the etherboot floppy image, use:

```
makerom
```

to choose "Intel Ethernet pro 100+" as the packet driver. And this utility will create two files called image.flo and image.rom. Once they are created they will never need to be upgraded. They are just small loaders for the system, linux kernel will be served from the NFS server. Examples of these two file are in /var/lib/diskless/boot/128.104.183.202.

To make the NFS server serving the linux kernel, a simple ftp server called "tftp" is needed. Please do:

```
apt-get install tftpd
```

and modify the file /etc/inetd.conf to change the default tftproot to /var/lib/diskless/boot. After you have the kernel ready (details in next section), you will need to copy the kernel to the boot directory at /var/lib/diskless/boot/<IP>, where IP is the IP address of the diskless node.

Then, if you are lucky, your diskless client should be able to boot.

2.2 Kernel preparation

Current kernel we are using is linux 2.2.18pre21. It is the consequence of my failing to improve the kernel performance of 2.2.17. More important, this version of kernel is the first version including many more new features that are just merged from kernel 2.4.x development. Those new features including NFS version 3 and P6 microcode, etc. The P6 microcode support is really nice, it allow us to update the processor's microcode every time we boot the machine(details later).

The kernel source is in /usr/src/linux. Two config files are prepared for server version kernel and diskless version kernel. They are server.conf and client.conf, please use:

```
make menuconfig
```

to use these two config files. After the menuconfig, to compile the server kernel:

```
make zImage
```

to compile the client kernel:

```
make bzImage
```

The reason for why the server kernel is smaller is because it is fully modularize. Only necessary modules are included in these two kernels.

After compilation, to install the server kernel, please:

[2]They are already in Beowulf-1:/root/debs/diskless.

```
cp arch/i386/zImage /boot/vmlinuz-2.2.18pre21
lilo
```

to install the client kernel, please:

```
cp arch/i386/bzImage /var/lib/diskless/boot/128.104.183.202/linux-2.2.18pre21
cd /var/lib/diskless/boot/128.104.183.202
./mknetbootimage
```

. After reboot, new kernel will be used.

Parallel software installation

To massively install all open source parallel computing software, use:

```
apt-get install task-parallel-computing-dev task-parallel-computing-node
```

These are two virtual packages to include all parallel computing development packages and cluster node runtime support packages. Note, you only need to install them on NFS server and master diskless node once. All shadow diskless nodes will have them automatically. "task-parallel-computing-node" is the runtime supporting library and binary for a cluster node. If you want to do the development, such as compiling a MPICH code, you will need "task-parallel-computing-dev" package.

Currently, MPICH 1.1.2, LAM2 6.3.2 and PVM 3.4.2 are installed. Only MPICH are fully configured and tested. The detailed performance of MPICH is presented in section 5. But both LAM2 and PVM are usable, we just did not have time to run example code of them.

To run LAM2:

```
lamboot -v /etc/lam/bhost.lam
mpirun_lam -np <#> <lam MPI program>
lamwipe -v /etc/lam/bhost.lam
```

To run PVM:

```
pvm <hostfile>
```

this will give you pvm console.

Based upon these three parallel framework, numbers of applications are also installed:

- Netpipe: A network performance tool. NetPIPE is a protocol independent performance tool that encapsulates the best of ttcp and netperf and visually represents the network performance under a variety of conditions.
- Blacs: Basic Linear Algebra Communications Subprograms. The BLACS project is an ongoing investigation whose purpose is to create a linear algebra oriented message passing interface that may be implemented efficiently and uniformly across a large range of distributed memory platforms.
- ScaLAPACK: ScaLAPACK is the parallel version of LAPACK.
- xmpi: GUI console of MPI_LAM2.
- xpvm: GUI console of PVM.

3 System Maintenance

3.1 NIS/YP and Autofs

Both NFS server and diskless nodes are configured with ECE NIS/YP services and kernel automounter. To do that:

```
apt-get install nis autofs
```

When prompted, enter the NIS/YP domain name: YP.ece.wisc.edu. Then modify /etc/auto.master, /etc/passwd, /etc/shadow, /etc/group to add a "+" character at the end of each file. Then restart autofs service by:

```
/etc/init.d/autofs restart
```

Then add tcsh and .setenv to /usr/local/bin by:

```
cd /usr/local/bin
ln -s /usr/bin/tcsh .
touch .setenv (or ftp from dolphin)
```

To enable ftp and user ftp login, do:

```
apt-get install wu-ftpd
vi /etc/shells (add /usr/local/bin/tcsh)
```

Then ECE users can login the cluster with their ECE account and their home automounted.

However, running MPI/parallel program from their own account is not recommended. This is because the remote home directory may be resided on a computer with slow disk or slow network, then the parallel programs run out of that remote home directory will be slow to load and slow to write their disk data. To address this issue a special disk pool is created in /var/autofs/misc/shared. Every user in users group can write to that disk pool and put the parallel programs there, and run from there. Because that directory is locally shared between cluster hosts, the file system IO speed is much faster than remote home directory. (At least for myself. :)

3.2 rdate and NTP

All hosts in the cluster are running NTP[3] to calibrate their clocks. To install:

```
apt-get rdate ntp
```

and enter the time servers: "time.nist.gov" and "time.mit.edu". After complete of the installation, use:

```
rdate time.nist.gov
```

to initiate the correct clock time.

3.3 ruptime

To monitoring the hosts in the cluster, ruptime daemon is used. To install:

```
apt-get install ruptime
```

then ruptime will show something like:

```
[beowulf-1] 38% ruptime
Eagle     up 2+10:05,  3 users, load 0.00, 0.00, 0.00
Falcon    up 14+23:14, 0 users, load 1.00, 1.00, 1.00
beowulf-1 up 3+08:30,  2 users, load 0.97, 0.53, 0.20
beowulf-2 up 20:43,    0 users, load 0.00, 0.00, 0.00
beowulf-3 up 20:43,    0 users, load 0.00, 0.00, 0.00
beowulf-4 up 20:43,    0 users, load 0.00, 0.00, 0.00
beowulf-5 up 20:43,    0 users, load 0.00, 0.00, 0.00
beowulf-6 up 20:43,    0 users, load 0.00, 0.00, 0.00
dolphin   up 23+10:58, 4 users, load 0.00, 0.02, 0.00
laser     up 18+07:22, 0 users, load 0.29, 0.42, 0.33
```

[3]Network Time Protocol.

3.4 reboot_all

To simplify the control over all diskless nodes. A small script is written and placed in /root/bin/reboot_all. In this script, all diskless nodes will be rebooted instantaneously. Only root on beowulf-1 is allowed to use this command.

3.5 software document

Thanks Debian, all software installed are with proper documents, they are accessible from:
http://beowulf-1.ece.wisc.edu/doc/

3.6 microcode update

A package called microcode-ctl is installed for automatically update P6 microcode if available. The start script is in:

```
/etc/init.d/microcode-ctl
```

And it is invoked every time system boot. If the microcode update in

```
/etc/microcode.dat
```

is newer than the current P6 microcode, the newer version will be uploaded to the processor, but it is not written permanently to the processor. Next time when system boot, it will be uploaded again. To check the result of the upload please:

```
dmesg
```

All of our current CPUs are with the newest microcode, 22June2000. Please keep a eye on
http://www.urbanmyth.org/microcode/
newest update of the microcode.

3.7 add shadow diskless node

A script called "new-beowulfs" is written and placed in /root/bin/new-beowulfs. Every time after the system admin updated the master diskless node which changes files in /etc or /var, please run the script. It will automatically propagate those changes to all the shadow diskless system.

Currently, only 4 shadow nodes are defined in /root/bin/new-beowulfs. If we want to added more shadow clients, please add its IP address into that file and run it. After that, a boot directory need to be added for the new nodes, simply:

```
cd /var/lib/diskless/boot
ln -s 128.104.183.202 <new IP>
cd /etc
vi bootptab (add a new line to define the MAC address of the new node.)
vi exports (add a group of new lines to define the shared directory for the new node. Please man expo
```

4 System Security

A cluster system is a tightly coupled group of hosts. "rsh" is used by parallel software to start remote processes. So the system are especially vulnerable if not carefully protected.

Debian is a highly secure system and by default some insecure services are turned off. While installing I turned them off for convenient access. Here is a list for how to turn them back off:

```
apt-get remove telnetd (only use ssh)
cd /etc
vi securetty (comment out remote ttys to disable root login remotely)
vi ftpusers (comment out root to disable root ftp)
```

5 Parallel Computing Environment on The Beowulf Cluster

Debian Linux distribution is a Beowulf-friendly Linux package, it provides MPI and PVM sub-packages to satisfy user's needs for workstation cluster computing. We installed two versions of MPI implementations from Debian 2.2: The MPICH 1.1.2 standard reference implementation and the LAM implementation. We also installed PVM as a reasonable alternative to MPI as most current MPI implementations don't support full-featured MPI-2 specification, therefore, cannot handle dynamic process management gracefully. We didn't install Condor due to the lack of time and its greedy diskspace consumption if in operation.

5.1 Communication Bandwidth Tests

We tested the communication bandwidth of our cluster with the stress test program and got the following performance graph (See Figure 2)

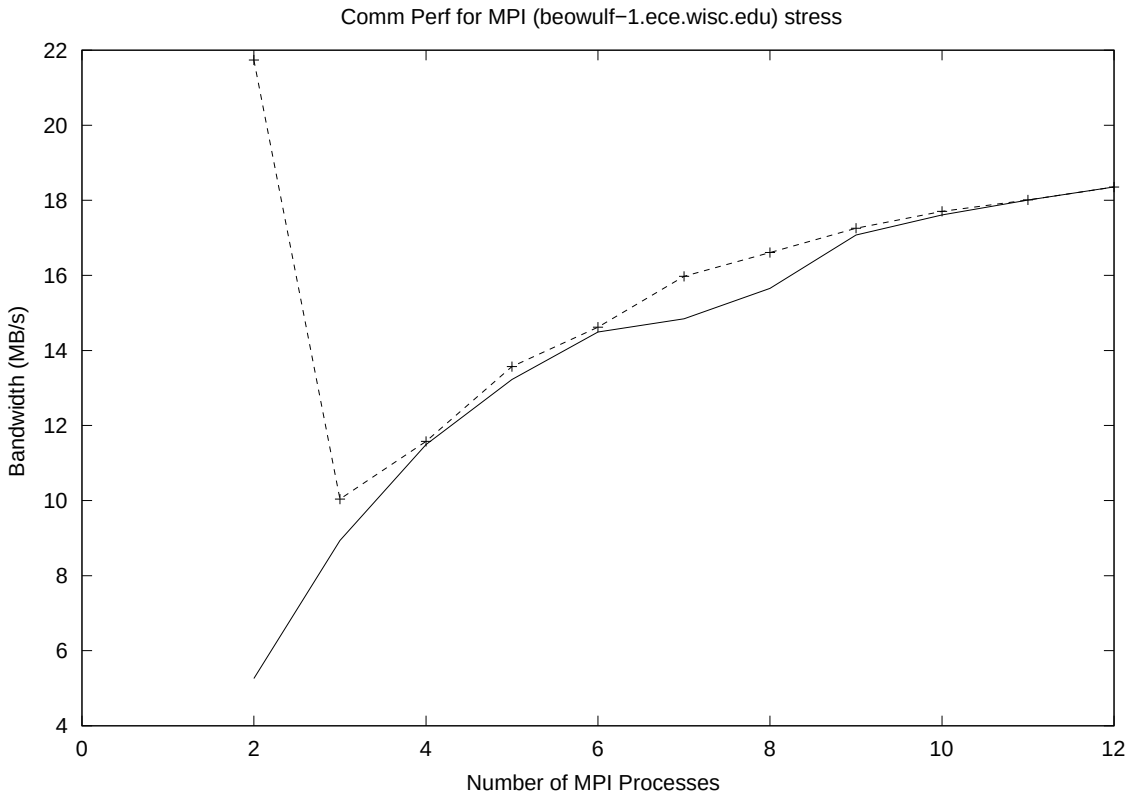

Figure 2: Bandwidth Test Results

MPI startup method:

```
mpirun -np (2-12) stress 102400 204800 3200
```

And the dotted line shows an alternative startup method with a user specified process group file to assign processes to hosts: we will refer to this startup mode as the p4pg mode/startup later on.

```
mpirun -p4pg [procgroupfile] stress 102400 204800 3200
```

Because each Beowulf node has two processors, our pocess group file tries to dispatch two processes to each node, if possible, in hope of utilizing data locality and/or better interprocess communication within the single node. A sample process group file for a 6-process task is given below. Note that the first column specifies which host to run the process followed in the same line and the second column specifies how many processes you want to add to that node. The order of the lines makes sense as for Beowulf clusters, MPICH use UNIX command 'rsh'

to dispatch tasks to remote nodes, therefore, waiting for one node to finish starting all processes assigned to it before issuing processes to other nodes would negatively impact the performance.

```
beowulf-1 0 /var/autofs/misc/shared/perftest/stress
beowulf-2 1 /var/autofs/misc/shared/perftest/stress
beowulf-3 1 /var/autofs/misc/shared/perftest/stress
beowulf-1 1 /var/autofs/misc/shared/perftest/stress
beowulf-2 1 /var/autofs/misc/shared/perftest/stress
beowulf-3 1 /var/autofs/misc/shared/perftest/stress
```

Our experiment results demonstrate different performance behaviors between the two different startup methods. p4pg startup, we beat the standard startup on points 2, 3, 7, 8. We would assume the reason for the 2-process p4pg startup's fancy 22.0MB/s bandwidth is due to the message bypassing inside the TCP/IP protocol stack. It might also explain the reason for points 3, 7 and 8. As far as points 11 and 12 are concerned, we basically did the same thing as the standard way. Hence, there should be no advantage and difference at all.

MPI specification provides an abstract communication interface to MPI programs and different MPI implementations working on different hardware platforms would map this abstract interface to the available underlying hardware communication mechanisms as efficiently as possible.

By default, each MPICH process running on a Beowulf cluster communicate with other MPICH processes within a certain communicator through TCP/IP connections. MPICH calls this kind of networked clusters as p4 device. We could choose shared memory mechanism for interprocess communication instead of TCP/IP network connection. Our intuition may support this strategy as each node in our Beowulf cluster is a 2-processor SMP system, shared memory may provide much higher performance. However, our experiments shows the other way by two facts. One is that the current Linux kernel doesn't support the mmap UNIX system call very well, therefore, the shared memory mechanism resorts to System V shared memory mechanism, which has a notoriously poor performance because of its mutex and other implementation mechanisms that have horrible overheads and seriously serilized operations. Another important fact is that we haven't figured out how to make MPICH support two modes of communication mechanisms simultaneously, with shared memory mechanism providing the fast communication channel inside each SMP node while at the same time with TCP/IP protocol stack supporting inter nodes network communications. Suppose we choose shared memory mechanism as the only mechanism for interprocess communication within the whole cluster system, the performance would be much poorer than TCP/IP alone because we have much more inter nodes commucnications than intra node communication in this situation. System V's shared memory mechanism works poorly compared with TCP/IP when handling interprocess communications across the 100Mbps LAN. Our experiments shows that shrecd memory mechanism alone is at least 10% slower than TCP/IP protocol alone. Hence, we have to choose TCP/IP as the only communication mechanism within our Beowulf cluster.SubsectionCommunication Latency Tests

We performed the latency tests using the tcomm test program. Similar to the bandwidth test, we performed two sets of tests, one with the standard startup method:

```
mpirun -np (2-12) tcomm
```

And another way is the p4pg mode by specifying user decided process group file.

```
mpirun -p4pg [processgroupfile] tcomm.
```

Not similar to the bandwidth test is that in this case, we cannot beat the standard startup by taking advantage of the message bypassing inside the TCP/IP stack. In this case, p4pg mode behaved rather unstable compared with the standard way.(Figure 3)

With the standard startup, the average rate maintained stable. However, the deviation of the links increases as the number of processes increases. We measure this by means of average number of links out of range. (See Fig. 4) It's worthy to mention that as the number of out-of-range links goes up, the extent of the performance difference between different communication links also goes up, which is not shown here.

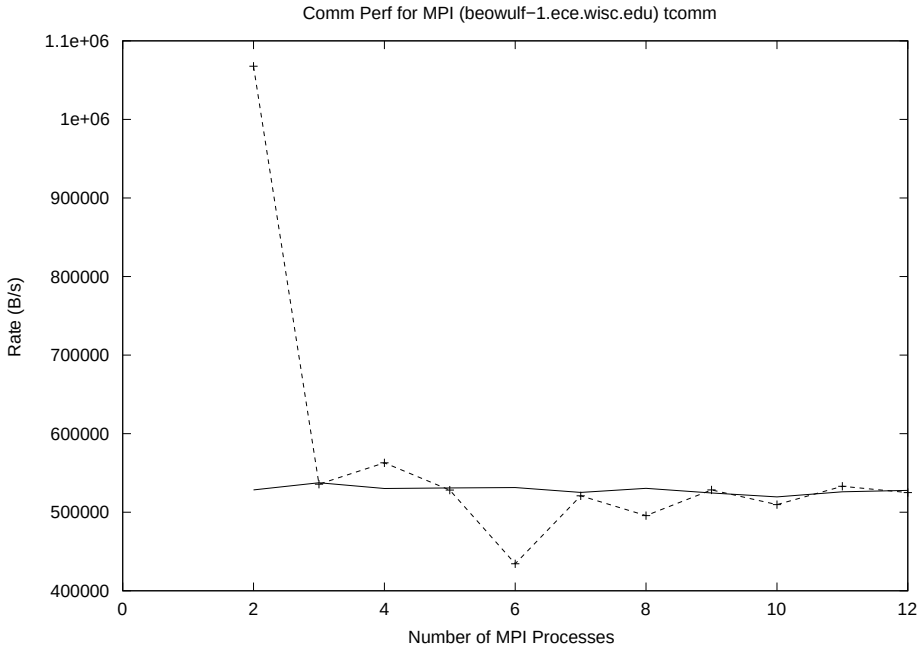

Figure 3: Latency Test Results

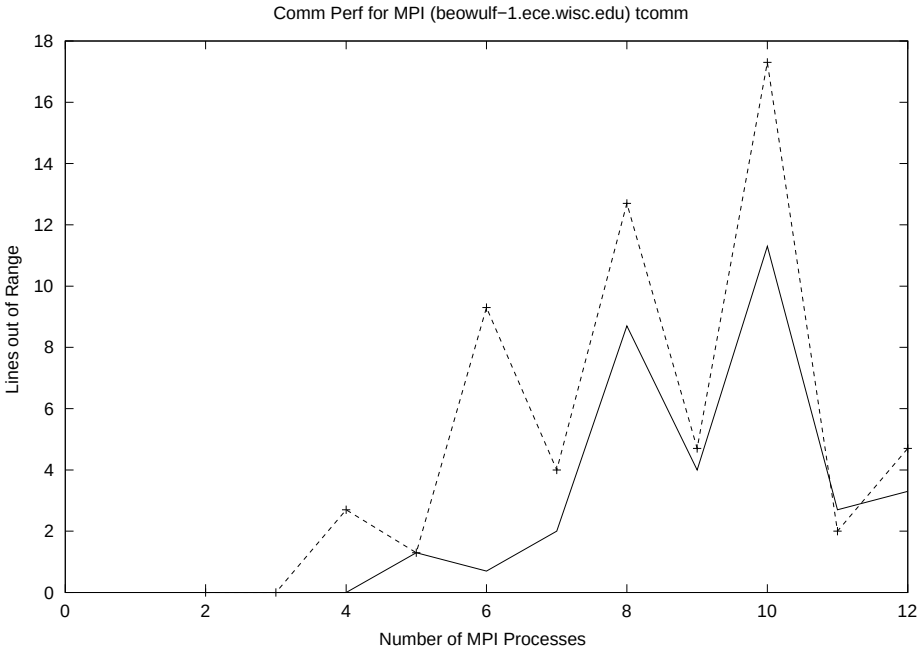

Figure 4: Deviation of the Latency.

5.2 Other Performance Tests

Other performance tests that we have performed including the mpptest. We did it both with the standard startup and the p4pg mode. For the mpptest with standard startup, the 11 performance curves, standing for 2-12 processes respectively, almost totally overlapped with each other(see Figure 5. But for the p4pg startup, the first curve has a exceptionally shorter times than all the other curves due to the fact that they actually communicate within the same SMP node.(See Figure 6) [END OF THE DOCUMENT]

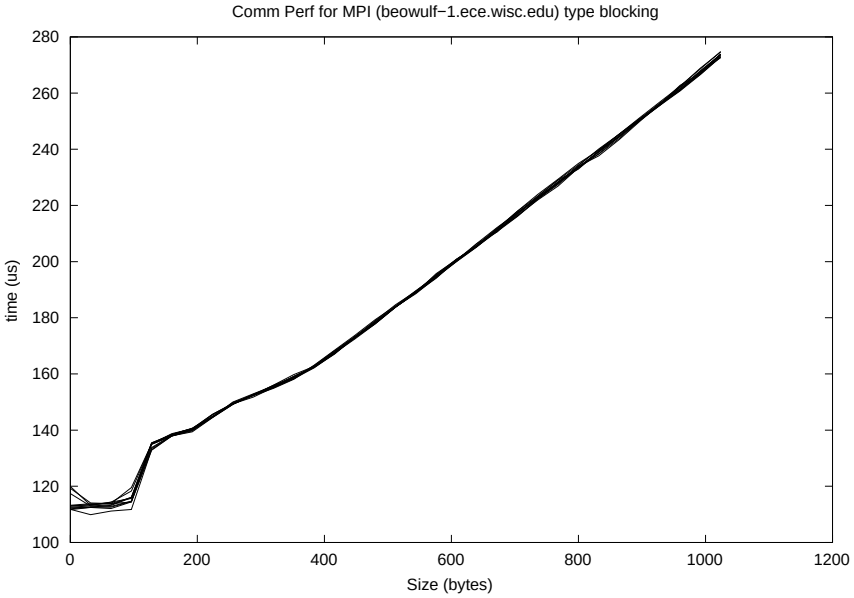

Figure 5: mpptest with standard startup.

The mpptest with our p4pg startup:

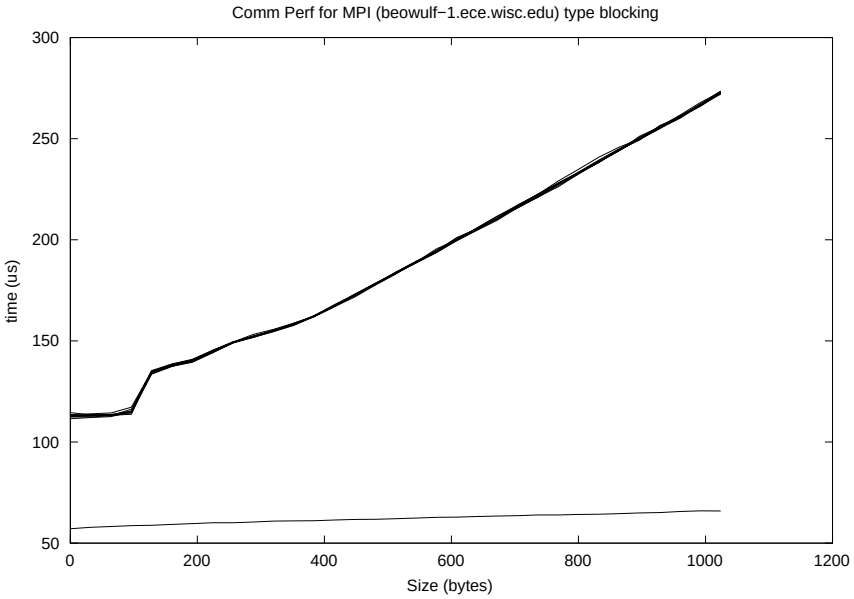

Figure 6: mpptest with p4pg startup