Stanford Exploration Project, Report 111, June 9, 2002, pages 401–??

Short Note

Cluster building and running at SEP

Robert G. Clapp and Paul Sava[1]

INTRODUCTION

SEP has always been interested in problems that stretch our computational resources. Over time the high-end computer at SEP has gone from array processors (Newkirk, 1977) to the Convex (Claerbout, 1985) to the CM5 (Biondi, 1991) to the Power Challenge to multi-processor Linux machines (Biondi et al., 1999), and in the last year, its first Linux Beowulf cluster.

This fall SEP decided to further expand its computer resources. We found the most cost effective option was to build our own Beowulf cluster. This paper attempts to summarize our cluster building experience and to explain how we program and manage our new cluster. In addition we discuss the current limitation of our setup and future plans to overcome some of these obstacles.

DESIGNING

When we first decided that we wanted to buy a new cluster we got quotes from several different vendors. These quotes varied up to 50% for the same configuration based upon the add-on features that each company provided. However, even the lowest figure was 30% more than what we calculated if we built the cluster ourselves. Once we decided that the 30% cost differential was significant enough to make building it ourselves worthwhile, we had to determine what exactly we wanted to build.

Processor benchmarking

Our first question was what type of processors to use. At the time there were three options: a Pentium 3 (P3), Pentium 4 (P4), and AMD Athlon. The P3 and AMD could be bought in a dual configuration for a reasonable price, while the P4 dual Xeon was prohibitively expensive. The cost of disk and memory along with space considerations made a dual system an attractive option. To test which configuration was the best for our current needs we built a single node

[1]**email:** bob@sep.stanford.edu,paul@sep.stanford.edu

of the P4 and AMD and compared it to our current P2 and P3 machines. For the P4 we used a 1.7 Ghz processor and Rambus memory. For the AMD we used a 1.2 GHz processor and DDR memory. We built a dual AMD (1.2 GHz) node and compared it to our current P2 and P3 configurations. Figure 1 shows the comparison of running a small wave equation migration program on each configuration. Assuming a little less than linear speed up with clock speed for the P3, we estimated that we would get approximately a 10% speed up with a 1.2 Ghz P3. The end result of this initial testing was that we felt that the high price of Rambus memory did

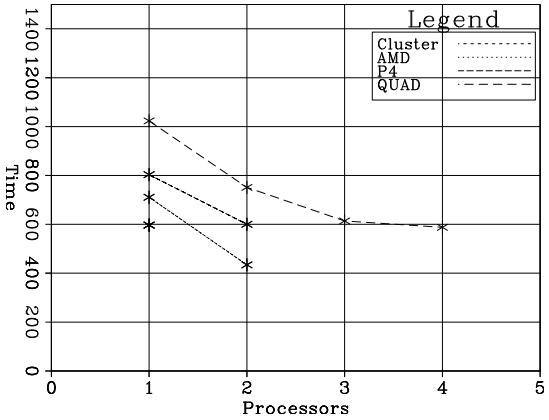


Figure 1: Benchmarking wave equation migration on quad Pentium 2, dual Pentium 3, Pentium 4, and dual AMD. The single '*' is for the Pentium 4. bob2-speed [NR]

not justify the speed improvement, therefore, we abandoned that option. The AMD and P3 were approximately the same cost (the P3 processor was more expensive but DDR memory was more expensive than SDRAM). We were leaning more towards the AMD build with the assumption that DDR memory prices would decrease as it became more standard when Intel made an announcement that they were going to release a new compiler for linux.

PGI vs. Intel

The Intel compiler was interesting because it seemed to promise a significant speed up compared to the PGI Fortran90 compiler. It was able to produce processor-specific instruction sets rather than general i486 instructions. In additon, it was supposed to be better at handling large memory problems than the PGI compiler. At the time we made our decision, we did not have a working version of SEPlib for the Intel compiler, so we decided to dumb down our tests to simple matrix multiplication. Figures 2 and 3 show the result of doing a real and complex matrix multiplication with a matrix of four million elements. Note how the Intel specific code is significantly faster than the corresponding PGF code. The speed advantage offered by the Intel compiler led us to choose the dual P3 option for our cluster.

Disk storage

Our problems tend to be not only computationally intensive but also large. As a result, disk space had to be worked into our design. The two most common approaches are either to put significant disk on each node and then create a virtual filesystem, using somthing like the Par-

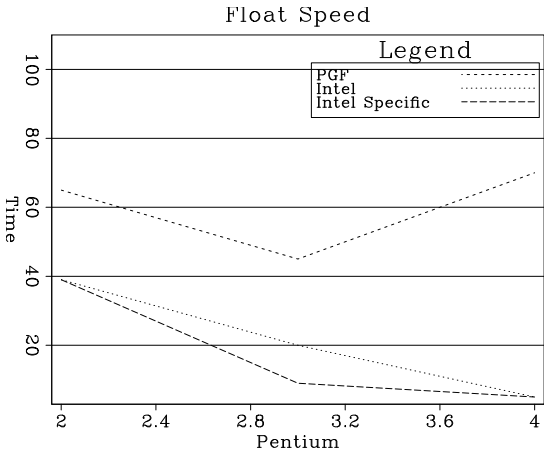


Figure 2: Speed comparison for matrix multiplication. The horizontal axis is machine (P2-550, P3-800, P4-1700). The different curves represent the PGF compiler, Intel compiler, and Intel with machine-specific instructions. Note the significant advantage of the Intel machine specific code. bob2-float-comp [NR]

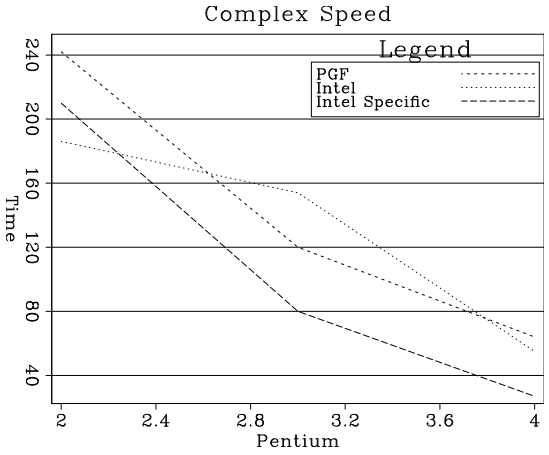


Figure 3: Speed comparison for complex matrix multiplication. The horizontal axis is machine (P2-550, P3-800, P4-1700). The different curves represent the PGF compiler, Intel compiler, and Intel with machine specific instructions. Note the significant advantage of the Intel machine specific code. bob2-complex-comp [NR]

allel Virtual File System (PVFS) [2] or creating a large disk server with a high speed connection to the cluster. Both approaches have drawbacks. The PVFS approach is dangerous because it means relying on fairly immature software. In addition, unless we wanted to use a Redundant Array of Inexpensive Disk (RAID) greater than zero (meaning we would have to mirror our data), we could run into problems whenever a node became unavailable. The disk server approach also has drawbacks. Many of our applications have significant IO requirements and a gigabit connection still has latency issues.

We decided to follow a model that allows the greatest flexibility in programming styles. Our solution was a combinitation of two approaches. We put in a large case:

- two dual process 1.2 GHz Athlon boards
- a gigabit network card
- a 100 megabit network card
- IDE raid control cards with 2.4 terabytes of disk.

The total cost of this unit was less than $9000. In addition, on each node we put a 60 gigabyte disk. For jobs that have significant IO requirements the local disk can be used. For large initial and final data, the disk server is available with a high speed connection.

BUILDING AND SETUP

Once we decided on our configuration we ordered the various parts (motherboard, processors, memory, disk drives, CDroms) from several different vendors. The actual cluster building took six students about $\frac{2}{3}$ of a day. Figures 4 through 6 show several stages in the building process. We chose motherboards which allow network boot but for our initial installation we installed off of a CDrom.

We used a HP Procurve switch to interconnect the nodes and our disk server. Each node uses `dhcp` [3] from our disk server for network configuration. We linked our current cluster to our old cluster using a single crossover cable.

We decided that a standard batch system such as the Portable Batch System [4] was inappropriate for our computer usage patterns. Batch systems are effective in insuring that CPUs are in constant use but they have several limitations that make them difficult to use at SEP. The most significant issue is that SEP problems generally take up significant disk space and can have high IO needs. With a batch system you have little to no control over which node your process starts on. As a result you can pay a high IO cost. The other problem with a batch system is that its design parameters don't address the type of problems we encounter in a research

[2] `http://parlweb.parl.clemson.edu/pvfs/index.html`
[3] `http://www.dhcp.org/`
[4] `http://www.openpbs.org/`

Figure 4: Cluster building. bob2-comp1 [NR]

Figure 5: Cluster building. bob2-comp2 [NR]

Figure 6: Cluster building. bob2-comp3 [NR]

environment. At best, a batch system determines the number of processes to use at start up. We occasionally have jobs that will take up to a week to run. A traditional batch system will either fill all available nodes with the job, making the machine unavailable for everyone else for a week or start it on such a few number of nodes that a week's job will take two months. Neither option is optimal.

To monitor the nodes we wrote a simple service that returns the result of the `ps` command when a certain port is accessed. This methodology proved more reliable than a simple `rsh` approach. From our entry point we periodically attempt to contact each machine and store load and usage information. For easier evaluation of this information we wrote a simple web interface.[5] Currently we limit computer usage by assigning separate accounts, but make the usage transparent by using the same `uid` for both SEP and cluster accounts.

For console access we debated whether to use a Rocket Port Serial Hub or a series of switches. Both solutions introduced significant cabling and additional costs. For now we are using the Virtual Network Computing (VNC) developed by ATT [6] for general console displays and attaching a monitor and keyboard when we experience problems with a node. So far this solution has proven effective but may not be optimal for a large system.

FUTURE

So far we have generally been happy with our new cluster. The speed up on large applications has been consistent with our benchmark results. We still see nodes occasionally hanging, but

[5] `http://sepwww.stanford.edu/sep/bob/sep.html`

[6] `http://www.uk.research.att.com/vnc/`

less so than with our previous cluster. In the future we plan to significantly expand our number of nodes, but we first need to deal with the problem of hanging nodes and find/develop a batch system that will meet our needs.

For nodes that die or hang we can, and do, use checkpoint systems, but these require significant coding by the programmer and are generally sub-optimal. A better solution seems to be to use/build upon one of the fault tolerant MPI versions. We should be able to easily build into/add on a batch system that will meet our needs.

We envision writting applications as a group of tasks (these could be a set of CMPS, a set of frequencies, or a set of shots for example). The batch system would:

- Initially send out the set of tasks to each available node.
- If a task is finished it would check to see if another process, with higher priority, needs the node. If it does the new process would get the node. If not a new task would be sent to the node.
- If a task does not finish it will be resent to a new processor.

This approach is similar to the models used by Seti@home [7]. It has the advantages that it is fault tolerant and allows dynamic reassignment of nodes based on need.

CONCLUSIONS

In this paper we summarized our computer building experience. We found the most cost effective processor was a dual P3 with SDRAM. We found that MPI provides the best scalability, but the standard implementation's lack of fault tolerance is problematic. In the future we see using/developing a fault tolerant MPI which allows for dynamic change in the number of nodes used by a given process.

REFERENCES

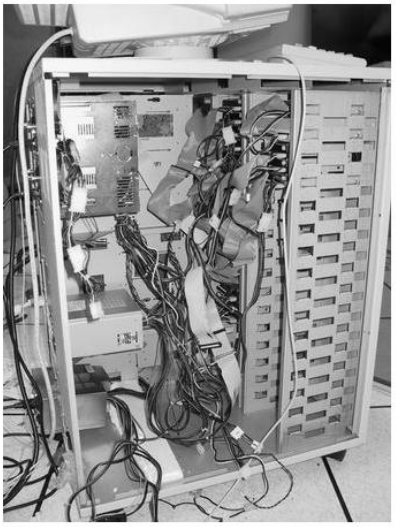
Biondi, B., Clapp, R. G., and Rickett, J., 1999, Testing linux multiprocessors for seismic imaging: SEP–**102**, 235–248.

Biondi, B., 1991, Wave equation algorithms on massively parallel computers: SEP–**70**, 59–72.

Claerbout, J. F., 1985, The Convex C-1 Computer: SEP–**44**, 173–180.

Newkirk, J., 1977, Installing an array processor: A progress report: SEP–**13**, 121.

[7] `http://setiathome.ssl.berkeley.edu/`




Antec
ATX