Undergraduate Experience in Clustering at the SC07 Cluster Challenge

Andrew Howard, Alexander Younts, Preston M. Smith, and Jeffery J. Evans

Purdue University, West Lafayette, IN

Abstract. The Cluster Challenge held at SC07 in Reno, Nevada provided a unique opportunity for undergraduate students to build, benchmark, and run real science applications on a state-of-the-art cluster of computers, while working with real-world constraints and interacting with experts in the high-performance computing field. After building a cluster, students ran the HPC Challenge benchmark and two days' worth of jobs for the applications POP, GAMESS, and POVRay. In this paper, students who participated in the Cluster Challenge describe the process for preparing for the competition, outline the work performed to understand complex real-world codes, and report on lessons learned while preparing for and competing in the Cluster Challenge.

1 Introduction

The area of high performance computing and clustering in general is an advanced field that many are taking a renewed interest in as problems become ever more difficult and data grows ever larger. Cluster computing on commodity hardware has been growing in popularity since the concept was first introduced by Beowulf [1] at NASA, and brings high performance computing to many people who could otherwise not have access to it. At the ACM Supercomputing '07 Conference, a unique opportunity was presented to university undergraduate students to study and practice high performance computing on a machine of their own design.

At Purdue University, a team was assembled of undergraduates from many related disciplines: Electrical and Computer Engineering Technology, Computer Science, and Electrical and Computer Engineering. Led by Purdue's Rosen Center for Advanced Computing and with the support of Purdue's College of Technology, the students enrolled in an existing course in high-performance cluster computing to prepare for the Cluster Challenge. Though a generous loan from Hewlett-Packard and support from AMD and Matrix Integration, we acquired a leading-edge cluster of computers to prepare on and use in the Challenge at SC07.

2 The Cluster Challenge

The Cluster Challenge invited teams of undergraduates to experience the challenges of cluster computing with several constraints:

- The cluster is limited to only a single 30 amp, 110 volt circuit, with a soft cap at 26 amps.
- At the start of the challenge, each team will run the HPC Challenge [2] benchmarks, and submit results to judges.
- Upon completion of the HPCC benchmark, teams receive data sets for POP, GAMESS, and POVray, and have 44 hours to complete as much of the workflow as possible.
- Points are issued for scores on 4 components of HPCC, for each data set completed, and 20 points of "judges discretion" were available for award by the panel of judges.

3 The Cluster and Preparation

HP provided us with a cluster of 13 ProLiant DL145 compute nodes, each with 2 sockets of dual-core Opteron 2216 CPUs and 4GB of RAM per node. The nodes were connected with Gigabit Ethernet for NFS traffic and a 4xDDR InfiniBand fabric for low-latency parallel communication. While the cluster was being arranged with HP, early possibilities of the rules included some consideration for performance per dollar, and we elected to keep price at a reasonable level by not using the lowest-power CPUs or loading up with huge amounts of memory.

In the lab section of our class, small teams of students set up each aspect of the system, from the CentOS 4 operating system, to an automated kickstart deployment, to batch systems (PBS and Condor), and compiler and MPI combinations.

Small teams of students were assigned an application, and spent much of their semester building, testing, and learning to understand the performance characteristics of their application. Each application was generally built with different combinations of compilers, optimization flags, MPI implementations, and in some cases, BLAS libraries. Shortly before departing for SC07, team members prepared papers as classwork on what they learned while preparing their application.

In this paper, we will present the results of preparing two of these applications: the Parallel Ocean Program (POP) and the Persistence of Vision Raytracer (POVray), and a summary of what we learned during the course of preparing for and competing in the SC07 Cluster Challenge.

4 Case Study: POP

4.1 Introduction

The Parallel Ocean Program (POP) [3] is an ocean circulation model developed by researchers at Los Alamos National Laboratory (LANL) in conjunction with the U.S. Department of Energy. Derived from the earlier models of Bryan, Cox, Semtner and Chervin, POP has been used for the ocean modeling portion of other climate models, such as the Community Climate Simulation

Model (CCSM) [4] from the University Corporation for Atmospheric Research (UCAR). This paper will focus on the LANL version of POP, even though the CCSM version of POP should perform similarly.

In this section we will look at the small-scale application and performance of POP. This also serves as a tutorial for getting POP up and running on an AMD system. Two test machines were used. The first was the HP cluster used for the Super Computing 2007 Cluster Challenge. The second machine was an SGI Altix running SuSE Enterprise Linux with 128 Intel Itanium processors and 512 gigabytes of memory.

4.2 Overview of POP

POP performs two different types of calculations: barotropic and baroclinic. Although understanding these algorithms isn't completely necessary, understanding the difference between these two calculations is important for analyzing POP's performance. Baroclinic calculations are three-dimensional integrations which take a significant amount of CPU time. Barotropic calculations, on the other hand, solve for surface pressure in a two-dimensional plane, making the barotropic calculation much faster than its three-dimensional counterpart. However, as we will see later in this paper, the barotropic calculation doesn't scale quite as well as baroclinic calculations.

Figure 1 shows how POP splits three-dimensional ocean data into a horizontal plane of points, known as blocks. POP forms groups of these blocks that are distributed among the nodes and CPUs. This idea is key to understanding how communication between nodes affects POP's performance.

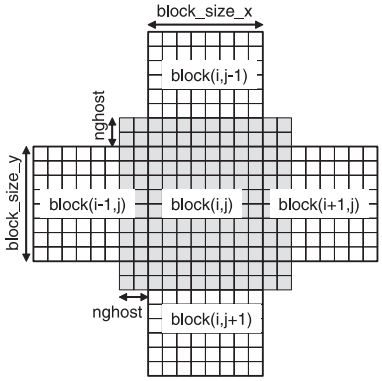

Fig. 1. A block in the two-dimensional decomposition of POP [5]

According to Kerbyson, et. al, [5], there are two main parallel activities in POP: boundary exchanges and global reductions. Boundary exchanges refer to the transfer of data at the boundaries of each of the blocks between nodes/CPUs.

Global reductions occur whenever POP requests a summary of all of the data among nodes (i.e. when time-averaged output file is created). While it seems POP isn't very communication intensive, it actually reaches these exchanges quite frequently. A lot happens during these exchanges, so it is important to have an interconnect fabric that can handle the loads. In the case of the HP test cluster, most of the communication took place over 20Gb InfiniBand.

Another important aspect of POP to consider is local I/O. The output data from the application can be reasonably large, growing at times to multiple gigabytes. Most of this data is appended to a growing file on the nodes. If local disk space is used on each node, this isn't much of a problem. However, if this disk I/O occurs on a network share, the network and disk arrays can become quickly saturated. It is a good idea to separate this disk traffic from the node intercommunication to prevent oversubscribing the network. The HP cluster solved this problem by using 1Gb ethernet for NFS traffic instead of the InfiniBand.

4.3 Compiling POP

POP has very few dependencies in order to be compiled. Since it can be run either serially or in parallel, all that is really needed is a compiler and the NetCDF libraries [6], along with the actual POP code from LANL [3]. However, since these calculations can take days to process serially, the use of a message passing interface (MPI) is preferred. For the HP machine, MVAPICH was used to take advantage of the InfiniBand fabric. MVAPICH was also used on the SGI Altix.

While any compiler supporting Fortran 90 can be used to compile POP, the PGI compiler was found to be best suited for the test clusters, based on information provided by AMD [7]. Instead of compiling POP directly with `pgf90`, the PGI compiler was used to compile both MVAPICH and the NetCDF libraries.

Once the NetCDF libraries and MPI program were compiled, POP was ready to be installed. The command `./setup_run_dir test` was used to create a POP executable directory named "test." Inside of the new directory, line 162 of `linux.gnu` file was modified in order to allow the PGI Fortran compiler to work properly. This change is shown in Appendix A. This file was also modified to point to the correct locations of MVAPICH and the NetCDF libraries. Next, the values of `max_blocks_clinic` and `max_blocks_tropic` were set to 92 in order to ensure each core would receive its own block. Before compiling, the `$ARCHDIR` variable had to be set to "linux" in order for make to work properly. Finally, POP was compiled by using the command `gmake`.

4.4 Running POP

POP is very flexible in customizing input files and parameters. Since there is plenty of documentation for using POP and customizing settings, this section will only serve as a brief introduction to using the test data set. Settings discussed here can be altered to fit any use of POP.

pop_in The file `pop_in` serves as the starting point for any POP run. This file acts as a pointer to all other POP input files, as well as specifying the parameters for POP's calculations. In order to analyze the performance of POP, only the fields `nprocs_clinic`, `nprocs_tropic`, and `lredirect_stdout` need to be modified. The first two fields are used to specify the number of processors available for each calculation. In the case of these tests, the number should be equal in order to allow both barotropic and baroclinic calculations to have equal resources. The last field tells POP to redirect the calculation details and run times to a specified file.

The `pop_in` file also needs to point to the transport and tavg files. These files set the starting point for the POP run. History and movie files can also be referenced to initialize POP, depending on the output desired.

NetCDF vs. Binary Output One of the major factors in large scale applications of POP is the writing of output files. The problem rests with the fact that NetCDF output does not support parallel I/O while binary output files do. Since the HP test cluster is only writing data from 44 CPUs at worst case, calculation performance actually *decreases* when using binary output rather than NetCDF output. Likewise, the Altix uses shared hard disks in RAID, eliminating the need for parallel I/O. Because of this, NetCDF output was used for all of the tests.

4.5 POP Performance

Since POP has already proven to scale well over a large number of nodes *(see [5])*, the tests were only run towards the upper bound of the cluster.

Performance on the HP Cluster Samples were taken from 24 and 32 CPU runs and the average total, baroclinic, and barotropic calculation times were calculated and are shown in Table 1.

Table 1. Calculation times for 24 and 32 CPU runs

	Baroclinic (s)	Barotropic (s)	Total (s)
24 CPUs	1.128	0.305	1.969
32 CPUs	0.327	0.052	0.444

The run times for the 32 CPU run were much better than the 24 CPU run. This is most likely due to the fact that POP was able to split the data into smaller pieces among all of the processors. Also notice that the total runtime wasn't affected by the fact that NetCDF output files were used instead of binary files. With a small number of nodes, parallel output does not have a significant effect on the final runtime.

Performance on the SGI Altix Run time was also measured on the SGI Altix using 25, 50, 64, and 65 processors. Figure 2 shows the breakdown of time spent on barotropic and baroclinic calculations, as well as the total combined time. The results were slightly surprising. Even though POP doesn't have to run on a square number of processors, the performance does seem to increase slightly when run on 25 or 64 processors. Figure 2 also shows that barotropic calculations don't scale very well and have a very minimal effect on the overall runtime.

On the other hand, baroclinic calculation times improve when run over a larger number of processors. At 64 cores, the total time spent on baroclinic calculations was 4.14 seconds. When the number of processors was increased to 65, the time actually increased to 4.37 seconds. While this isn't a major increase, it is important to keep in mind that the baroclinic time at 25 processors was only 4.7 seconds. This means that the 65 core run only provides an overall performance improvement of 1.07 instead of the 1.13 seen at 64 cores.

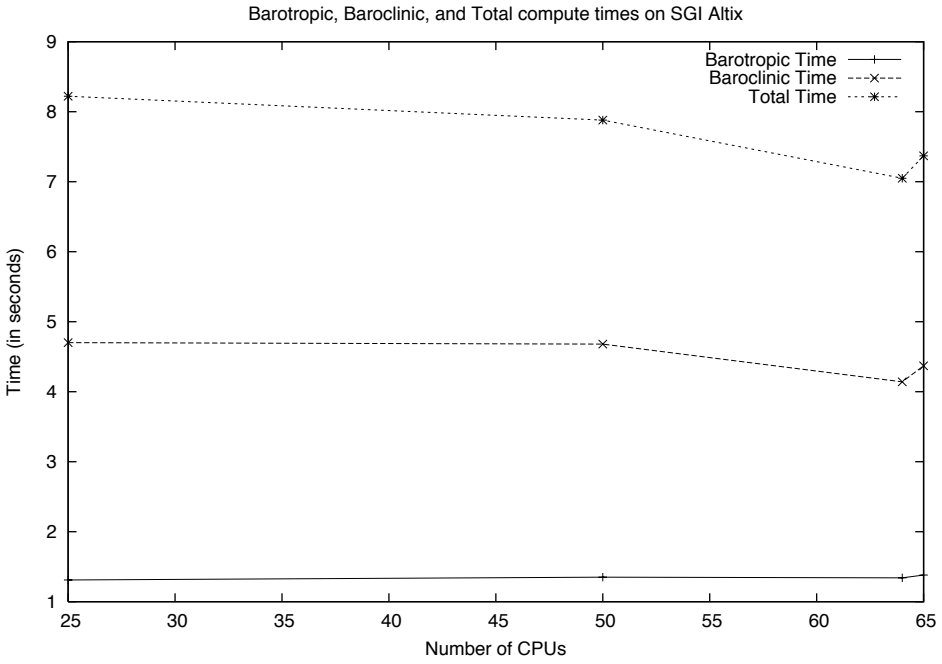

Fig. 2. Average POP runtimes on SGI Altix

4.6 Visualizing POP output

Since POP is used in a variety of climate simulation packages, there isn't one standard program used to visualize its output. Because of this, it becomes very

hard to find a program that *will* visualize POP output without requiring other parts of a climate simulation. One program that does visualize POP's output is Ferret, from the National Oceanic and Atmospheric Administration (NOAA). While the program itself is very basic, it allows for very strict control over what is visualized. A key thing to remember when using Ferret is that it handles NetCDF output very well, but visualizing ASCII and binary output is a little harder. Therefore, it is recommended that NetCDF output is used.

After Ferret has been installed using the installation guide on the Ferret website, the program can be launched simply by typing "ferret" on the command line. Then the NetCDF output file from POP can be loaded with the command `set data <filename>`. This will load all of the variables and their values from the POP output file. If the command `show data` is typed, the program will show all of the variables currently set, including the number of time steps in each variable.

There are two main types of plots that are useful when plotting POP output: shade and vector. Shade plots are used to map variables such as surface height and salinity, i.e. variables that don't necessarily have directional attributes. Vector plots are best suited for viewing current and wind data, i.e. variables that do have directional attributes. The great thing about Ferret is it allows you to pick and choose how to plot what data, and also allows for combining both shade and vector plots. An example of a shade plot is shown in Figure 3.

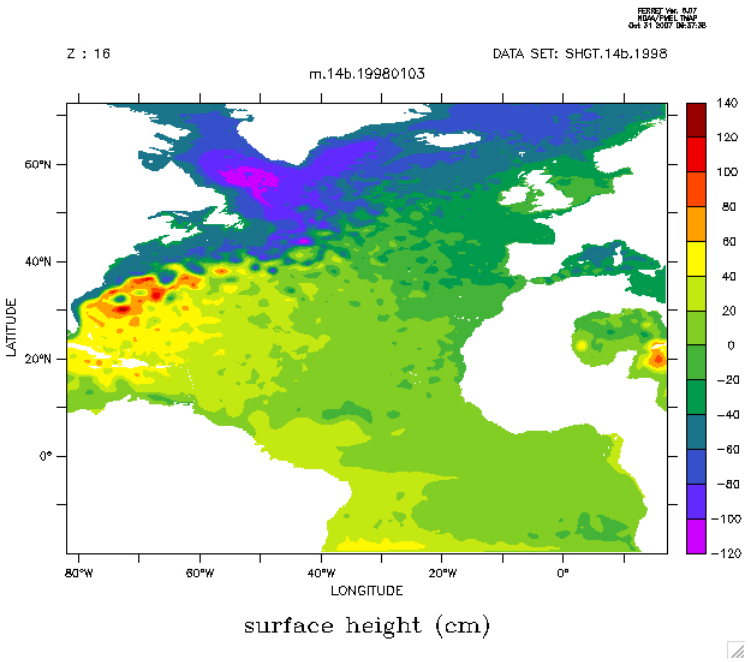

Fig. 3. Example of Ferret's "shade" plot

Suppose there is a variable in the POP output called "SHGT" which contains data on the surface height of the water over time (k). This would be best suited for a shade plot. To plot the data at the first time step (k=1), simply type the command `shade/k=1 shgt`. Note that this command will only open a window with he visualized data; it will not write it to a file. Instead, the command `shade/k=1 shgt; file=output.gif` should be used to write the data to a GIF image file.

The "repeat" command can then be used to create multiple GIF files over time. To do this, simply type the command:

```
\texttt{repeat/k=1:12(shade shgt; FRAME/file=output.gif)}
```

Ferret will plot the variable "SHGT" for twelve time steps, create a sequentially numbered GIF file for each step.

A benefit of using Ferret is that it can be run using a script file with an extension ".jnl". This script file is created each time Ferret is run, but it can also be written in a text editor. The JNL file simply contains a list of commands for Ferret to run. To run using the script file, simply type:

```
\texttt{ferret~-script~<scriptfile>}
```

4.7 Conclusion

This paper serves as a rudimentary introduction to using the Parallel Ocean Program and Ferret. While it isn't a complete summary of what the programs can do, it does cover the basics needed to understand how each program works.

Since a large amount of parallel I/O isn't needed, the runtime improves significantly by increasing the number of CPUs used. The tests could have been improved by increasing the complexity of the input data, as well as using half of the CPUs in each node rather than using all four processors. Scaling 24 CPUs over 12 nodes instead of 6 would have a significant effect on inter-node communication, since the nodes would have to use boundary communication more often and global reduction would take longer.

Also, the overall runtime of processors should be kept to an even number, since this makes it easier for POP to split the topographical data into blocks. An even more ideal situation is to use a square number of cores. While this doesn't increase performance a ton on short runs, long runs would see a significant improvement in runtime.

5 Case Study: POVray and Condor

The Persistence of Vision Raytracer, or POV-ray [8], is a software package that is widely used to create very vidid images that can look almost exactly like real life photographs. It does this by rendering images in much the same way regular human vision works. Normally, beams of light come from a light source, a lightbulb or the Sun, then strike an object and reflect a colored beam of light into our

eyes; many beams of colored light hitting our eyes allow our brains to construct an image. POV-ray does this same process, except in reverse.

In real life, a lot of beams of light get made and make wild journeys but never get collected by a human eye. If POV-ray were to act like the real world, enormous amounts of computing time would be spent making and tracing the lives of beams that would never get seen. Because of this, POV-ray creates beams that start their lives from the point of view of a camera and then get traced back to whatever objects and light sources would be in the way to modify the beam. Every pixel in the final output image from POV-ray is a beam that got sent out into a scene and ended up as a color value.

POV-ray can take a scene and create virtually any sized image, although the number of pixels in the final output corresponds directly to how long POV-ray takes to render it. The problem of taking time to compute can be compounded further if a user is creating an animation that consists of many different images. In an animation, the objects, the camera positions, the light sources or all three can all be changing. This means POV-ray must render each pixel in the output single frames by generating a separate beam. This means a lot of computing time and a lot of resources.

Condor Condor [9] is a resource manager for very diverse systems, created by the University of Wisconsin. It solves the problem of having a lot of computer systems used for a variety of tasks but that spend some amount of time not performing their main function. Condor makes use of these wasted resources by matching work to the resources available.

A classic example of Condor is to convert a regular set of office computers into a computation grid at night when all the office employees are at home. Condor will collect a queue of jobs with specific requirements. Then, as computation resources join or leave the grid, Condor will control what jobs are running. Given the overall leaps of speed in today's processors, Condor produces a cluster of machines that can take advantage of massive amounts of idle compute cycles.

Purdue University has a Condor pool of approximately 7,000 machines. A large numbers of these machines are workstations in computer labs. Only a few number of computer labs stay open twenty-four hours a day, which leaves a lot of computers sitting idling by waiting for users to show up again in the morning. This provides a wonderful increase in computing power in the late evening and early morning hours. As well, if a machine goes idle for some length of time, Condor will make use of these off peak hours to fit in even more computation time.

POV-ray and Condor The version of POV-ray used for this paper was the most recent available and was version 3.6. This version does not include any message passing capabilities which makes running POV-ray on a cluster to be fairly wasteful unless there is a way to parallelize the work in another way. Thankfully, when creating an animation, there are a lot of frames to be rendered and they are all generated independently. This makes the job of rendering an

animation a perfect fit for running on a cluster or on a machine with multiple processors as users can render multiple frames at a time on different nodes or processes.

With the potentially sporadic nature of resource availability in a Condor pool, jobs generally cannot be multi-node or multi-processor as any one node might be reclaimed by its owner. This makes Condor suitable for a type of job that has a relatively short life time, or that can checkpoint, and does not require multiple nodes or processors. (This is assuming machines in a Condor pool are not completely dedicated to running Condor jobs. Having dedicated machines for use by Condor is possible.)

POV-ray fits the requirements of a typical Condor job perfectly. It generally takes on the order of less then ten minutes to compute a frame, and by using the "+C" flag when invoking POV-ray, it will continue rendering a frame if it has been interrupted. This behavior is a nearly perfect fit for a Condor job in an environment where resources are not dedicated. Also, because jobs are generally short and can be restarted without loosing much work, they are perfect for backfilling against other computing tasks.

If an office worker decides to take a short, thirty minute lunch, then Condor can make use of most of the free time on their workstation by running a set of POV-ray renderings. This allows for a computer to stay near full utilization all the time, given that there is work to be done and the infrastructure can support the number of nodes currently in Condor.

Keeping expensive computing resources always busy is a essential. Once time passes, you can never retrieve those lost cycles. Putting every cycle to good use and wasting none is a very important requirement when running work units for the Cluster Challenge.

Using POV-ray and Condor Using Condor can be very similar to submitting jobs using the Portable Batch Scheduler. You create a submission file and run a command to insert your job(s) into the system. Here is a sample file used to generate the data for this paper:

```
universe = vanilla
Executable = /opt/povray-3.6.1-gcc/bin
/povray
Log = $(Process).log
Output = $(Process).out
Error = $(Process).err
Arguments = +W800 +H640 -Ipovcatray.pov
 -O$(Process).png
transfer_input_files = povcatray.pov
transfer_files = always
Queue 100
```

In the example above, one hundred Condor jobs are created in the Vanilla Universe that use "povray" as the executable and with the requirement that all

needed files are transferred by Condor to the target machine before execution begins.

The example Condor job will render the same image one hundred times over. To render a video frame by frame, some more options in the "arguments" are needed. POV-ray includes two options, a start frame and an end frame. Adding "+SF$(Process) +EF$(Process)" will render the frame corresponding to the job ID. To render a movie with three thousand frames, you just need to spawn three thousand Condor jobs.

Once a submission file is created, running "condor_submit input.file" will have Condor parse the file and spawn the correct number of jobs with the correct requirements. Using "condor_status" can list the progress of the jobs in condor, and once all the jobs in the rendering are complete, all the output files will be in the directory where the "condor_submit" command was run.

5.1 Opportunities for Tuning

There are many tunable parameters between Condor and POV-ray. There are three major phases that POV-ray goes through during a rendering run: parsing the scene file, rendering the frame, and saving its output. The two major functions that get performed during these times are file input and output and computation. Computation and parsing require the most time, second to file input and output.

Choosing a particular compiler is very important as that can reduce the total amount of time required for POV-ray to complete its heaviest tasks. The compilers that were available to test were those from the Gnu Compiler Collection, the Intel®Compilers, and the Portland Group's compilers.

The second opportunity for optimizing a system to run POV-ray is how the system deals with file input and output. As the cluster had a shared file system, its possible to store the input scene files and the output renderings from POV-ray there. This requires that all file operations happen across the network. Another possible way of dealing with the movement of data is to allow Condor to manage this aspect. Condor will transfer files before it begins executing user code and then bring the output files back to the submission machine after a job exists.

5.2 Predicted Outcomes

POV-ray automatically chooses very conservative optimizations to use while building binaries. It defaults to only passing the "-O3" optimization flag to most compilers, which should make stable binaries and give a good look at how each compiler tested does with generating code for our AMD Opteron^{TM}processors. Both the Intel®Compilers and the Portland Group Compilers include special optimizations that could benefit POV-ray, but only common features were tested. The Portland Group compilers have a reputation of producing high speed code for the AMD platform. The Intel®Compilers and the Gnu Compiler Collection were included because of their popularity and accessibility.

The best file performance should come from using Condor to manage the transfer of data. Doing some basic benchmarks with IOZone [10] on the file systems of the master and computer nodes, the master node achieves around 70MB per second and the compute nodes get around 45MB per second. The disk subsystem from the master node is shared, and if multiple jobs are running at a time, then they must share the total amount of disk bandwidth on the master node. The bulk transfer mechanism of Condor should be better than the constant stream of NFS traffic.

5.3 Compilers

To compare the different compilers, a sample image was chosen from the standard POV-ray installation. Then, that image was render 100 times and the times averaged. The results can be found in Figure 4.

The Gnu Compiler Collection scored the best run times of all three compilers with a average time of 50.34 seconds and a maximum run time of less than the Portland Group's minimum run time.

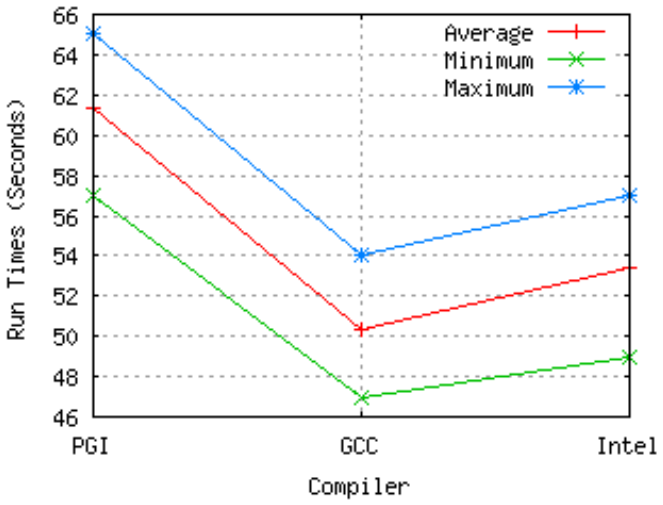

Fig. 4. Compiler Results

5.4 File System Performance

To compare different file transfer mechanisms, a sample image was chosen from the standard POV-ray installation. To simulate different animation rendering requirements, this image was rendered at two different resolutions. One image

size was rendered at 720x480 (Standard Definition) and the other image was rendered at 1920x1080 (High Definition). These resolutions represent the possible output formats that an animator might want for display.

For the lower resolution image, better performance was seen by using Condor's transfer mechanism, 18.76 seconds compared to 19.63 seconds from Figure 5. For higher resolution images, better performance was seen by using NFS, 73.03 seconds to 73.26 from Figure 6, although the different here was relatively small. Condor's minimum and maximum transfer times for the large format image were both higher than NFS's times.

The output files sizes were one megabyte for the smaller image, and three megabytes for the larger image.

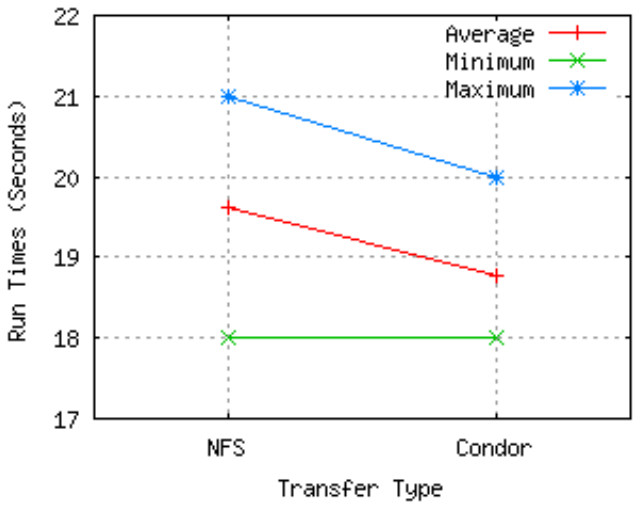

Fig. 5. Standard Frame Results

5.5 Conclusion

The best combination of using readily available tools for running POV-ray on a cluster appears to be the Gnu Compiler Collection to build the POV-ray binaries and then different file handling techniques depending on output file size required.

A possible explanation for the Gnu Compiler Collection doing remarkably well is that AMD and Novell have been contributing directly to the project [11]. Some of their optimizations specifically help mathematical operations happen more efficiently on the processor. The direct involvement of AMD in the development of Gnu Compiler Collection and not testing special features from the Portland Group Compilers is probably what gave the Gnu Compiler Collection

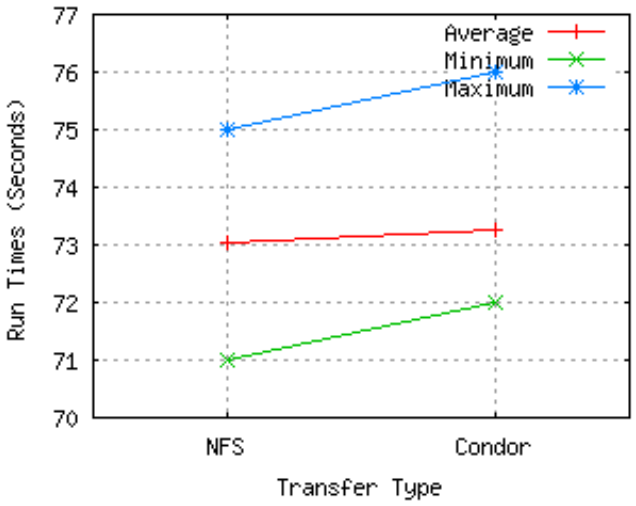

Fig. 6. Large Frame Results

the best performance on the system. Before turning on the advanced optimizations used by the Portland Group, POV-ray should be tested to ensure proper operation as the manual does not mention that it has been extensively tested with these compilers.

For smaller images, local file operations would be faster because the latency between the compute node and the master node would be larger than going to local disk. After a certain point, the increase speed of the disk system in the master node appears to outweigh the increase in bandwidth. Given the number of instances potentially running at a time on the test cluster, 48, it might be safer to use the Condor transfer mechanism to protect against swamping the file server during a time sensitive animation rendering.

6 Lessons Learned at the Challenge

6.1 Power Constraints

Finding cluster resources at the undergraduate level is generally not something that can be done. The best that can be obtained what is found in a dumpster and pieced together to look more like a machine from Frankenstein than something that should sit in a data center. Given the amount of working compute nodes a dumpster can hold, powering such a beast is usually not a problem after most machines have been melded to form working hardware. At the Cluster Challenge, however, a major resource limitation was the power budget. Having to power the nodes and networking equipment on 26 amps was challenging. When we arrived

at the competition, we also learned that our 26 amp power budget would be split across two circuits limited to 13 amps each.

To solve the problem of having two different circuits to run our equipment on, the team choose to remove one node from the cluster so as to consistently stay under the 13 amp requirement on both circuits. With an additional node on one of the circuits, power load would idle under the limit but quickly go over when jobs started to run. There was still extra capacity on the other circuit, but the monitoring software would not combine the readings together.

6.2 Adapting at the Competition

The first problem the team ran into after the start of the competition was recognized after analyzing the GAMESS input data sets. Our GAMESS expert, David Matthews, quickly saw that our nodes did not have enough memory in them to contain a working copy of the data set. Our initial goal of running all three codes at once on various partitions of the machine would not be possible because GAMESS required many gigabytes of shared memory, which would require the use of most of the cluster to have enough memory.

Since we had several spare nodes and one node that was not usable due to the power requirements, we transplanted memory from the inoperable nodes into working nodes. This allowed GAMESS to run without requiring the exclusive use of the cluster. In retrospect, having more memory per node would have been to our advantage, especially with GAMESS.

A couple days into the Cluster Challenge, after everything was in full swing, a completely unexpected event occurred: the power to the entire SC07 floor was lost. Several teams worked very hard the next few hours bringing their machines back to life after a variety of system failures. Beyond the obvious lost productivity of jobs running during the power loss, one team lost a node and another team had a node stuck in its power on process which prevented them from running jobs for a while. The Purdue team fared well after the power loss. The cluster, at the time of the event, was only running POVray jobs which easily handled the interruption with a very minimal loss of work. The Condor scheduling software did get temporarily stuck coming back to life, but it took a simple command to get it rescheduling jobs. Overall, the power outage was a good real world test of the cluster systems on the floor because it challenged the team's overall system design and knowledge, not just their knowledge of the three main applications.

The third and biggest challenge for the Purdue team was the loss of stability on the machine being used as the master node for the cluster because of problems with our RAID array. During the middle of the second night of the challenge, the system would experience unusually high load as processes on the master node became stuck in disk wait state if too many file I/O operations were happening. Troubleshooting the problem was difficult, at first glance, we suspected a problem with the POP application, then as perhaps just a NFS problem. In the end, the stability of the master node in general was blamed, and to work around this, we pulled all the files off of the dying system onto a regular compute node. Then, a configuration change was made to the entire cluster to pull shared storage from

that machine and use the head node for very light duty as just as a login and job scheduling machine.

6.3 Hardware Differences

Except for the Purdue team, all of the teams at the Cluster Challenge used processors from Intel, most of which had 4 cores per CPU socket. Purdue's dual-core AMD processors appeared, from a core-count perspective, to put our system at a disadvantage, with only half as many processor cores than some of the other teams. However, in practice, this did not turn out to be a handicap. If we did not have the RAID system failure, which cost several hours of productivity, we are confident would have been close to completing all the data sets or even finishing them all. The winning team completed all the data sets but there were several teams that did not finish all the data sets due to more drastic software and hardware failures.

The performance of the HP/AMD cluster in HPCC was a positive, despite our core count disadvantage: our teammates who were the HPCC experts sufficiently tuned the benchmark to achieve 83% of peak performance, which was in our opinion, excellent.

Our experience should show that having the greatest hardware available does not ensure a victory. Knowing the applications and system engineering better than anyone else can go a long way to providing the edge needed to secure a winning place. In fact, the winning team from the University of Alberta did not have the fastest CPUs or the most cores, but were very well prepared.

6.4 Learning Applications

We had the hardest time with GAMESS. Interestingly, we were unable to identify a single scientist at Purdue who ran GAMESS. Purdue's TeraDRE project [12], and an active community of computational climatologists at Purdue made expertise in POVray and POP much more accessible to the team. Conversations with the winning Alberta team showed the opposite: GAMESS expertise was plentiful but climatologists were scarce at their institution.

Finding real world expertise to aid in learning the applications and having test data sets of a similar scale to those we encountered in the challenge to run with the applications before the competition are important to properly learn the applications and perform well at the event.

6.5 Experience at the Conference

Attending a large conference for the first time can be quite an experience and being able to actively participate in an aspect of a major conference is even a better opportunity. The team worked in shifts around the clock, which left some people not working during the day. This allowed team members to attend technical seminars, presentations and a chance to go around the show floor meeting

vendors and seeing the greatest new products coming on the market. At the team's booth, many people stopped to see what the challenge was about, how Purdue was doing, and to see what the individual team members thought of the industry. Many Purdue alumni made it a special point to came by the booth at least once during the competition to see how their alma mater was faring and meet the undergraduates on the team. Attending Supercomputing was an amazing opportunity for the entire team.

Acknowledgments

The authors would like to thank Hewlett-Packard, AMD, and Matrix Integration for their support of the SC07 Cluster Challenge.

At Purdue: the Rosen Center for Advanced Computing, Information Technology at Purdue, and the Colleges of Technology, Science and Engineering.

The rest of the Cluster Challenge Team and classmates in ECET 581Y: David Matthews, Richard Miller, Josh Wildey, Trent Nelson, Donghan Ryu, Taesuk Yoon, Noel Diaz, Glen Jungels, John Burgess, and Michael Whitfield.

References

1. T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages I:11–14, Oconomowoc, WI, 1995.
2. Jack J Dongarra and Piotr Luszczek. *Introduction to the HPCChallenge Benchmark Suite*. Ft. Belvoir Defense Technical Information Center, December 2004.
3. Los Alamos National Laboratory. *Reference Manual for The Parallel Ocean Program (POP)*, May 2002.
4. William D. Collins, Cecilia M. Bitz, Maurice L. Blackmon, Gordon B. Bonan, Christopher S. Bretherton, James A. Carton, Ping Chang, Scott C. Doney, James J. Hack, Thomas B. Henderson, Jeffrey T. Kiehl, William G. Large, Daniel S. McKenna, Benjamin D. Santer, and Richard D. Smith. The community climate system model version 3 (ccsm3). *Journal of Climate*, 19(11):2122–2143, 2006.
5. Darren J. Kerbyson and Philip W. Jones. A performance model of the parallel ocean program. *Internation Journal of High Performance Computer Applications*, 2005.
6. Russ Rew, Glenn Davis, Steve Emmerson, Harvey Davies, and Ed Hartne. *The NetCDF Users Guide*. Unidata Program Center, July 2007.
7. Pgi compilers and tools.
8. Pov-ray: The persistence of vision raytracer.
9. Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.
10. Iozone filesystem benchmark.
11. Gnu toolset optimized for amd.
12. S. Lee Gooding, Laura Arns, Preston Smith, and Jenett Tillotson. Implementation of a distributed rendering environment for the TeraGrid. In *Challenges of Large Applications in Distributed Environments, 2006 IEEE*, pages 13–21, June 2006.