

SC07 Cluster Challenge: A Student Perspective

Dustin Leverman, Doug Smith, Brian Cairns,
John French, Tyler Knappe, Mike Levy

University of Colorado, Boulder
Dustin.Leverman@colorado.edu

Abstract. This paper examines the design and performance of a cluster constructed by a team of undergraduate students for the SC07 Cluster Challenge. The objective of the Cluster Challenge is to engineer a system that operates within a strict power envelope while maximizes performance on a set of prescribed benchmarks. A description of the hardware and software configuration is presented with emphasis on the challenges encountered in optimizing the benchmarks and the techniques employed to stay within the power budget. The paper distills several strategies for success from our experiences that apply to competitions such as the SC07 Cluster Challenge as well as to organizations undertaking general cluster architecture design and implementation.

1 Introduction

The proliferation of Linux clusters has made the field of supercomputing more accessible than ever before. This is due to increased innovation in hardware and software technologies as well as decreasing costs of commodity computational hardware available from a wide array of vendors. Linux clustering has developed to the point where a small team of undergraduate students can build and operate a system on their own.

In the SC07 Cluster Challenge competition, the objective was to construct a cluster that could complete the prescribed four benchmarks within a 48 hour timeframe while staying within a given power budget. Six teams participated in the competition: The University of Colorado, Purdue University, Indiana University, University of Alberta, Stony Brook University, and National Tsing Hua University. Each team came with their own hardware and software platforms provided by a wide range of vendors.

Many of the challenges faced by the Cluster Challenge teams are common issues faced everyday in cluster implementations. Tasks such as designing a cluster, configuring high performance interconnects, installing library software, and benchmarking applications are essential not only for the competition, but as part of everyday high-performance computing operations. In this paper, we overview the underlying hardware and software components, the challenges of compiling, running,

and optimizing the four target models (HPCC, POP, GAMESS, and POV-Ray), and examine the challenges of staying within a strict power budget.

2 About the Cluster Challenge

The Cluster Challenge, first introduced in the 2007 Supercomputing conference, is an engineering competition where teams consisting of six undergraduates design, build and optimize a Linux cluster under the guidance of a faculty advisor with optional financial support from vendor partners. Clusters may be constructed using any architecture that can run the benchmark suite, but the cluster may use no more than 26 Amps and must fit in a single 42U rack. Each cluster must provide a minimum of 50 gigabytes of available storage, a single DVD drive, and a 42" flat panel monitor to place on the front table for displaying model output during the competition. The clusters may be constructed and benchmarked before the competition. However, once the competition starts, the cluster hardware configuration may not be changed. Each team is given a 48 hour period to complete the benchmark suite.

3 Hardware & Software

Our vendor partners were Aspen Systems and QLogic. Aspen Systems provided a full-height APC rack, HP Procurve Gigabit Ethernet Switch, and ten compute nodes; QLogic provided the Silverstorm Infiniband switch and host adapters. Each compute node contained the following components:

- Intel S5000 Motherboard
- Intel Quad-Core 5355 Xeon Processors (2 per node)
- 8 gigabytes of fully buffered DIMM memory
- Single on-board gigabit Ethernet
- QLogic QLE 7140 4x SDR Infiniband interconnect (2 per node)

Aspen Systems provided us with ten nodes. Our preliminary analysis suggested that we would be able to compete with between six and eight nodes within our power constraints, allowing us to retain two nodes as spares. However, due to actual power consumption, we were only able to compete with six nodes.

The Infiniband switch was used for inter-node communication by MPI, and the Ethernet switch was used for the management network. Using two Infiniband cards per node increased overall performance by improving the balance between the CPU, RAM, and interconnect. Fully-buffered RAM was used to provide consistent power consumption regardless of memory utilization.

The software stack consisted of CentOS Linux, PathScale Fortran and C/C++ compilers, Intel 9.1 C/C++ and Fortran compilers, and the combination of the Torque 2.3 resource manager and the Maui 3.2.6 scheduler. Other software required for

running the benchmarks included NetCDF 3.6.2, Intel MKL 9.1 (in particular, BLAS), and MVAPICH 1.2.7.

4 Benchmarking

At the competition, the teams were given 48 hours to complete the four prescribed benchmarks: HPCC, POP, GAMESS, and POV-Ray. Each team had the opportunity to optimize the benchmarks *before* the beginning of the cluster challenge. However, the final performance results from each benchmark were gathered during the time-limited competitive period. Each team was required to complete the HPCC benchmark before being given the input data sets for the remaining benchmarks.

4.1 HPCC

HPCC [3] is a benchmarking suite that is composed of 7 individual benchmarks, each measuring different aspects of the system: HPL, STREAM, Random Access, PTRANS, FFTE, DGEMM, and the `b_eff` latency and bandwidth tests. Most of these benchmarks are run as provided, but HPL requires a significant amount of tuning to produce optimized results. HPL measures how fast the system can solve a set of linear equations, and forms the basis for the popular Top 500 Supercomputer Sites list [8]. We built the HPCC suite with the optimized Intel math libraries. On HPCC only, we ran the Intel compiler because it showed an on average 30% improvement over the Pathscale compiler.

Optimizing HPCC / HPL

The basic install of HPL is not configured to run on a system that has several processors per node or a large amount of RAM. There are 4 main variables that need to be changed in the HPL configuration to optimize the benchmark to produce the highest possible results. These include: block size (NB), problem size (N), and the size of the process grid (PxQ) representing the number of nodes and tasks per node. In the case of our competition cluster, we ran several different tests to identify the optimal configuration. For example, we would adjust the equation used to find N to determine how changing the amount of RAM used would affect performance.

To experimentally determine the initial block size, we used Jack Dongarra's recommended calculation for setting the problem size [4]:

$$N = \sqrt{M/8} * .8$$

In this equation, M is the total memory in bytes in the entire cluster, divided by 8 to produce the number of double-precision numbers that can be stored in the HPL coefficient matrix, and multiplied by 80% to reserve some memory for the operating system. In our case:

$$N = \sqrt{(8GB * 7)/8} * .8 = 67731$$

We tested for bottlenecks by modifying the configuration file over multiple runs to include different numbers of nodes. For example, if we had a bottleneck in the interconnect we would have seen a decreasing marginal return by adding more nodes. Initially, we thought that we would be able to run with 7 nodes, so we tuned for this configuration. Later, we realized that 7 nodes took us over our power budget, so we were required to drop down to 6 nodes. The following two tables summarize our results from the WR11C2R4 portion of the HPCC benchmark for both node counts:

Table 1. Initial tuning results from varying block size (NB) with a constant problem size (N).

Case Number	PxQ	NB	N	# of Nodes	Speed (WR11C2R4)
1	7x8	80	67731	7	256.2 Gflops
2	7x8	120	67731	7	278.1 Gflops
3	7x8	160	67731	7	281.8 Gflops
4	7x8	200	67731	7	283.6 Gflops
5	7x8	240	67731	7	274.0 Gflops

*Note- N in previous example was derived from the N formula given in the above paragraph

Table 2. Initial tuning results from varying the problem size (N) while holding the block size (NB) constant.

Case Number	PxQ	NB	N	# of Nodes	Speed (WR11C2R4)
1	6x8	200	40000	6	228.8 Gflops
2	6x8	200	50000	6	234.4 Gflops
3	6x8	200	60000	6	237.7 Gflops
4	6x8	200	70000	6	245.4 Gflops
5	6x8	200	71000	6	245.2 Gflops
6	6x8	200	72000	6	243.9 Gflops

The results in Table 1 show that with both the problem size and the number of nodes fixed, a block size of 200 yields the highest performance of 283.6 Gflops. We then held the block size of 200 as fixed, and P and Q were set to 6 and 8 due to the power-limit of 6 nodes (with 8 cores per node). Since the optimized values for PxQ and NB were already found, N was left to optimize. Several tests were run and ultimately an N of 70000 produced the best result.

HPCC Results

The evaluation points in the HPCC section of the Cluster Challenge were divided into two parts: results from an un-optimized run of HPL, and the results of an optimized run of HPL. The un-optimized run is the distribution's default values, and are shown with our result in Table 3. This table also displays the results from our previously detailed optimized configuration.

Table 3. Submitted results for the HPCC HPL WR11C2R4 test.

	HPCC HPL Test	N	NB	P	Q	Time (s)	Gflops
Optimized	WR11C2R4	70000	200	6	8	926.23	246.9
Un-optimized	WR11C2R4	1000	80	2	2	0.05	14.03

During the competition, we completed the HPL runs in about 15 minutes. With strict time constraints it is essential to prepare an optimization strategy in advance, thus allowing for more time to work on the other models, considering that HPCC had to be run first.

4.2 POP

The Parallel Ocean Program (POP) [5] is an ocean model from Los Alamos National Laboratory that is used to simulate ocean currents as they evolve over time allowing for changes such as temperature and height. This software package is useful for benchmarking purposes because it is memory and interconnect intensive. It works by solving three-dimensional equations representing hydrostatic and Boussinesq approximations for fluid motion on a sphere. POP uses MPI for parallel communication and requires the Network Common Data Format (NetCDF) for file I/O. After building NetCDF and confirming that the package was fully functioning we ensured that POP could link with it.

Challenges / Solutions

The main challenges for POP were to optimize it to obtain the best performance from our compute nodes and interconnect. Although we initially wanted to use the Intel compiler, we discovered that it caused issues with our InfiniPath drivers. Instead, we used the PathScale compiler suite. The compiler suite includes a set of optimization tools to benchmark different compiler flags, which we used to eventually settle on the combination that resulted in the best performance. We also found that the performance with this model reaches a peak at five nodes. If we go over this many nodes, our interconnect traffic increases substantially making the model run slower.

POP Results

After switching compilers and finding that there was a sweet spot in performance in regards to the number of nodes used, we were able to get the maximum speed out

of this benchmark. Thanks to the compiler flag optimization program integrated into the Pathscale compiler, we were able to use the compiler to increase performance. In the end we were able to complete all of the models, while only using 3.75 hours of the 48-hour time window. The average runtime for each of the runs was about 2696.48 seconds.

4.3 GAMESS

The General Atomic and Molecular Electronic Structure System (GAMESS) [2] is an ab-initio quantum chemistry simulation application. Instead of MPI, GAMESS uses its own Distributed Data Interface (DDI) for message passing. Building GAMESS presented the challenge of organizing the modular build, modifying the settings on the scheduler, and modifying the build, link, and run scripts. Several issues with configuring and running GAMESS were encountered during the competition that suggest improvements to be made in future competitions.

Challenges and Solutions

The build of GAMESS was challenging in itself. The compilation, link and run scripts all have hard-coded directories left in place by the original developers. Finding and changing all of these according to the cluster's particular architecture proved to be somewhat frustrating.

Once GAMESS had been built, running a job with a data set proved to be error prone. The System V shared memory variable had to be increased to provide DDI with the necessary resources. This error was identified early, as test datasets would not run without an increased amount of shared memory. The System V shared memory was increased from the command line of each node individually shortly before the competition began. However, due to a power failure on the conference floor during the competition, the cluster suffered a hard reboot, resetting the System V shared memory to the insufficient CentOS default. Unfortunately, the System V memory issue had been diagnosed and addressed by a single teammate who was not available immediately after the power failure. As a result, the GAMESS jobs were delayed in favor of POP and POVRay jobs, until the particular teammate returned to the competition area. Such a situation could be prevented in the future by deploying a team knowledge base, which would serve as a dynamic representation of the changes made to the cluster. A Wiki could provide this functionality with little time devoted to setup.

In addition, the MPI quiescence detection setting had to be disabled to prevent the scheduler from killing the job while DDI handled message passing. As DDI handled the inter-process communication between the GAMESS processes, the MPI job launcher saw little activity. After some default time of no activity, the MPI monitor would inform the scheduler that the job was quiescent (it was not making any progress) and should be killed. This error appeared during the competition. It was not detected earlier because the test datasets did not use the resources of the cluster for a sufficiently long period. This was a simple problem to fix by adding '-q 0' to the

mpirun command in the run script. This issue highlights the importance of thoroughly testing the cluster, for all applications, both in terms of use of resources and duration, before the competition.

The setup completed during the competition successfully ran the majority of the GAMESS competition jobs. However, an unsolved problem with the setup was forcing GAMESS to use all 8 CPUs per node. At the competition it appeared that GAMESS would only use 4 CPUs per node for process computation, reserving the other 4 per node as data servers to handle DDI. This may have significantly reduced the cluster's computational potential for all of the GAMESS datasets. From the perspective of the scheduler, it appeared that GAMESS was only utilizing half of the CPUs on the cluster, which was highly frustrating. One of the configurations considered prior to the competition was testing the GAMESS DDI implementation against MPI. Due to the difficulty of configuring GAMESS for MPI, which is explicitly addressed by the GAMESS manual, no such test was performed before the competition due to the constraints on time (the competition was going to start the next day)[7].

Results

GAMESS took the most time to run out of the 4 benchmarking suite. It was the last model we ran, and by the end of the competition we were only able to complete four of the five test runs. The results were displayed on the team's main screen in the competition area using wxMacMolPlt [1]. Below are some of the images generated by wxMacMolPlt:

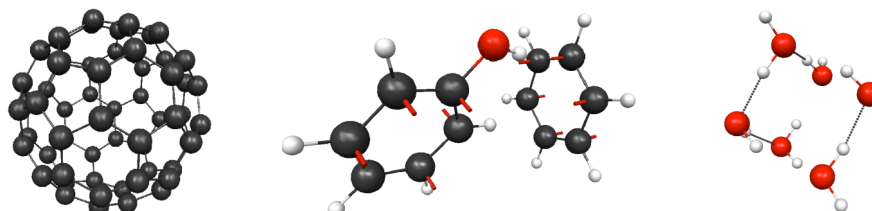


Figure 1. Example output visualizations from wxMacMolPlt.

Table 4. Performance results from the GAMESS runs.

Dataset	# of Procs	Clock Time	Processor Time	Processor Usage
1	24	11005.6	8259.2	75.05%
3	24	12060.1	10182.2	84.43%
4	24	19064.3	8868.4	46.52%
6	24	23854.2	23386.3	98.04%

The fifth and final dataset of the competition ran from the completion of the fourth dataset until the end of the competition, a period of about 23 hours, and was not

completed. In the haste to return the cluster back to Aspen Systems, no post mortem on the fifth data set was ever done.

4.4 POV-Ray

The Persistence of Vision Raytracer (POV-Ray) [6] is an open-source program used to create high quality computer-generated imagery using a special graphical technique called ray tracing. Ray tracing is computationally expensive compared to more traditional rendering techniques because it traces individual rays of light from the light source to the virtual camera rather than using an ambient light source. POV-Ray 3.6, the stable version used for the competition, is a single-threaded application that does not support symmetric multiprocessing (SMP) or MPI. Although there is an MPI version of POV-Ray, it was not stable at the time of the competition, so it was not used.

Challenges/Solutions

Rendering is a problem that is particularly well suited to parallelism, particularly when animation is involved. Because animations contain hundreds or thousands of frames, and because frames are generally computationally independent from each other, it is possible to divide the problem on a frame-by-frame basis and render different frames on different nodes. To get this to work we could divide the animation frames up beforehand and create render scripts for each CPU core; however, this is inefficient because different portions of the animation have different levels of computational complexity. This imbalance causes some cores to finish their jobs prematurely, resulting in them sitting idle until the animation has been fully rendered. The solution that we finally decided on relied on our job scheduler, Torque/Maui. In general, the scheduler is designed to handle large jobs, keeping the cluster busy by filling the available cores based on the resources that each job requests, such as the rendering of the individual frames in our animation. To utilize the scheduler, we wrote a render script that takes two parameters: the name of the POV-Ray animation, and the frame to render. The render script invokes POV-Ray to render the correct frame to an appropriately named bitmap. We then wrote a second job submission script, which also takes two parameters: the name of the POV-Ray animation to render, and the number of frames (n) to render. The submission script submits n jobs to the scheduler; each job consists of invoking the render script with the correct (incrementing) frame number and the correct animation name.

After the submission script was executed, the scheduler handled the distribution of frames to the nodes in the cluster. This provided for nearly 100% utilization. In addition, because we also used the Torque/Maui scheduler for our other jobs, it was possible to “backfill” any unused resources with POV-Ray frames when the cluster would otherwise be sitting idle. Under this environment, individual POV-Ray frame renderings were used to keep all nodes and all cores busy. By using the convert command in the ImageMagick suite of tools, the individual video frames could be

combined into one mpeg file. We were able to render all of the datasets, completing this section of the competition.

5 Power Challenges

With such strict power constraints, we needed to consider many system components to make sure that we would stay inside our power budget.

5.1 RAM

A cluster architect is faced with a broad variety of choices in hardware. Since some of the benchmarks are memory intensive, we chose memory that performs well and has low power consumption. Typical system memory consumes increasing power as utilization increases and can contribute a significant amount to the overall power requirements. To address this issue, we chose fully-buffered DDR RAM. Although this type of RAM consumes much more power while idle than traditional memory, under high RAM utilization the power requirements are stable and requires less power than un-buffered RAM. By using less power from our RAM when running the models, we were able to get the most performance from our power budget.

5.2 Balancing Power

Each team was limited to two power distribution units (PDUs). Each of these circuits had a maximum draw of 15 Amps at 120 volts, but an alarm would sound if the consumption of either circuit went over 13 Amps. The challenge was to spread the power use as evenly as possible between the two circuits. Initially, we put 4 nodes on one circuit and the remaining 3 nodes, Infiniband switch, and Ethernet switch on the other circuit. This strategy spread out the power evenly at idle, but when a job was running the circuit with 4 nodes would go over 13 Amps. Because of this, we were forced to drop down to 3 nodes per circuit and balance the two switches between each circuit. The program that really highlights the issues with the power budget was the HPCC benchmark. It used the most RAM and CPU, and also generated the peak power load. Since we were not allowed to change hardware once the competition started, the specifics of the power budget and circuit distribution reduced our expected node count by one, which significantly reduced the overall performance of the cluster by approximately 14%. Given a different power distribution, but still restricting the peak consumption to 26 Amps, we would have been able to use all 7 nodes and achieve higher performance.

6 Conclusions

The Cluster Challenge does not provide a ranking of the participants, only identifying the overall winning team, which in this case was the team that finished all the benchmarks, Alberta. However, we feel that our team was competitive. We finished all but one of the model runs in GAMESS and completed all other benchmarks. We came away from the competition encouraged about our performance and believe that our experiences will help the University of Colorado's team next year.

The power envelope and circuit distribution were the limiting factors for our cluster's size and resulting performance. The HPCC benchmark itself constrained our submitted design to six nodes, even though the other benchmarks were consistently run under the power budget for seven nodes. Given a chance to fully revise our design, we would look at a denser configuration of cores and evaluate the capabilities of the lower power processors. We would also attempt to reduce the overall number of in-chassis power supplies to reduce the overhead of the power losses of the power supply units themselves.

The other major contributing factor to our results was a relatively poor scheduling of the application runs. We started at the top of the list of model runs and ran through them in a sequential manner. We may have ran the GAMESS application earlier and, used POV-Ray to backfill un-used processors to take advantage of unused CPUs. As previously described, GAMESS was difficult to build and run optimally, and our results could have benefited from additional time and effort spent on its configuration and tuning prior to the competition.

During the competition, the venue experienced an un-planned power failure. This added a real-world aspect that we did not anticipate. The power failure cost us several hours to get back up and running, as we had to diagnose a hardware problem when the power came back on line. Torture testing our cluster by un-expectedly turning off the power to both the full system as well as individual components will be a part of our preparation for next year. Also, such an emergency highlights the necessity of having documented team knowledge easily available to all team members throughout the competition. A knowledge base, probably in the form of a Wiki, will be a key tactic for future CU Cluster Challenge teams.

By the end of the challenge, our team was able to complete all of the HPCC, POV-Ray, and POP benchmarks. As mentioned previously, one GAMESS benchmark was not completed in time. From optimizing POP using a different compiler, finding the soft spots in our hardware by optimizing HPCC, learning the importance of scheduling with GAMESS and POV-Ray, and balancing power to get the most out of the budget, we now have good foundation of understanding that will help us to be even more successful at next year's Cluster Challenge.

7 Acknowledgements

We would like to thank the following people and corporations for providing support and making the Cluster Challenge possible: Brent Gorda (LLNL), ACM/IEEE,

Merrill Lynch, Western Gecco, and Chevron for sponsoring the 2007 SC07 Cluster Challenge; Steve Spring, Lonnie Maynard, and Jaime McIntyre of Aspen Systems for sponsoring our team; Xiao-Chuan Cai of the University of Colorado Computer Science Department and Dr. Webster Cash from the University of Colorado Physics Department for financial contributions; and Michael Oberg, Matthew Woitaszek, and Henry Tufo for assistance with writing this LCI paper.

References

1. Brent Bode, wxMacMolPlt, 2007, <http://www.scl.ameslab.gov/~brett/MacMolPlt/>
2. General Atomic and Molecular Electronic Structure System (GAMESS), <http://www.msg.ameslab.gov/GAMESS/>
3. High Performance Computing Challenge Benchmark (HPCC), <http://icl.cs.utk.edu/hpcc/>
4. Jack Dongarra, Frequently Asked Questions on the Linpack Benchmark, "For HPL What problem size N Should I run?", 05/08/2007, http://www.netlib.org/utk/people/JackDongarra/faq-linpack.html#_For_HPL_What_problem%20size%20N%20should
5. Parallel Ocean Program (POP), <http://climate.lanl.gov/Models/POP/>
6. Persistence of Vision Raytracer (POV-Ray), <http://www.povray.org/>
7. The Gordon Group, GAMESS documentation, 2007, <http://www.msg.ameslab.gov/GAMESS/documentation.html>
8. Top 500 Supercomputer Sites, <http://top500.org/>