

# Bringing Disruptive Technology to Competition

Alexander Younts<sup>†</sup>, Andrew Howard<sup>‡</sup>,

Preston M. Smith<sup>§</sup>, Jeffrey J. Evans<sup>‡</sup>

<sup>†</sup>Department of Computer Science,

<sup>‡</sup>Department of Electrical and Computer Engineering Technology,

<sup>§</sup>Rosen Center for Advanced Computing

Purdue University, West Lafayette, IN

Email: {ayounts,ahoward,psmith,jje}@purdue.edu

**Abstract**—In November 2008, the second annual Cluster Challenge competition was held in Austin, Texas at the 2008 Supercomputing conference. Students from Purdue University made up one of seven teams of undergraduates to compete in the competition. Wanting a change of pace from the commodity hardware of the first challenge, Purdue teamed up with their vendor SiCortex, Inc. Students were then given the task of learning scientific applications POY, OpenFOAM, RAXML, WPP, and GAMESS, along with the HPC challenge benchmark suite. In this paper, students from the Purdue Cluster Challenge team discuss the work done in preparation of the competition, along with strategies and their effect on the outcome of the Cluster Challenge.

## I. INTRODUCTION

Starting during the early days of computing, there has been a demand for increased performance. At first these performance boosts were achieved through faster clock speeds and larger memory systems. In recent years, the trend has moved to multicore computing and, most recently, a “green” low-power computing movement.

Low power systems are increasingly becoming an option for HPC facilities for a variety of reasons:

- Electrical power constraints: caps on the amount of power available to the data center, as racks and racks of traditional cluster systems with faster clock speeds are added to the facilities.
- Cooling constraints and costs: computers need to be kept cool in order to operate properly. For every extra watt needed to power computing, up to another watt is required for cooling. Cooling equipment often takes up as much space as the computers themselves.
- Space constraints: data centers are simply running out of floor space as racks of cluster systems and the necessary cooling equipment proliferate.

Low-power computing architectures often use slower clock-rate CPUs, trading speed of individual CPUs in favor of a larger total quantity of CPUs. Also, a low-power system may have less memory than commodity hardware, to save power from having fewer RAM chips. A high-speed interconnect may also make the system better suited for communication-bound applications that may not necessarily require high clock-speed processors.

With the need for low-power “green” computing in mind, the Cluster Challenge provides a great platform for pitting

different configurations of machines against each other with one limitation: power.

## II. CLUSTER CHALLENGE

According to the SC08 Cluster Challenge rules: “The Cluster Challenge will showcase the amazing power of clusters and the ability to harness open source software to solve interesting and important problems. Teams will compete in real time on the exhibit floor, where they will run a workload of real-world problems on clusters of their own design. The winning team will be chosen based on workload accomplished, benchmark performance and overall knowledge and presentation.”

Each team provides a cluster of any design, but constrained only in that it must run on two 13A circuits. Team Purdue arrived at the challenge with one SC1458 system, partially populated with 6 CPU modules, for a total of 962 processor cores.

Teams then have 48 hours to benchmark their cluster and run a multitude of scientific application datasets. For the 2008 Cluster Challenge, these applications included WPP, POY, RAXML, GAMESS, and OpenFOAM. Each of these applications have varying degrees of scalability and demand rigorous testing to determine the ideal scheduling algorithm for the cluster.

## III. THE CLUSTER

Purdue University began working with Cluster challenge vendor SiCortex, Inc. in early 2008, when a SC5238 equipped with 3,240 processor cores was delivered to the Rosen Center of Advanced Computing (RCAC). A fully populated SC5832 has 5,832 cores. Researchers from several scientific domains have been using the machine to explore its “green” computing and code development and scaling virtues. Soon after delivery Purdue and SiCortex agreed to partner for the SC08 cluster challenge. The machine for the challenge was essentially a SC648, a single-rack product equipped with a slightly modified power distribution scheme (to meet cluster challenge requirements) and additional nodes, for a total of 162 nodes (972 cores).

### A. Node Architecture

Class and lab time was devoted to learning about the SiCortex cluster architecture. Each node “chip” consists of

six (6) low-power 64-bit MIPS cores clocked at 500MHz. Each core has a six-stage pipeline and can provide in-order execution of up to two instructions per cycle for a peak floating-point rate of 1 GFLOPs. As shown in Figure 1 the node chip also includes the L1 and L2 caches. Each core has its own 32KB L1 instruction and 32KB L1 data cache and a 256KB segment of the L2 cache. an interleaved memory controller for each DIMM, a DMA engine and interconnect fabric switch, and a PCI Express interface which is enabled only if external I/O devices are used. One of the key points stressed in our HPC class was the fact that the fabric switch and DMA engine connect directly to L2 cache. Bypassing main memory this way significantly improves communication latency and provides a tighter coupling between computation and communication.

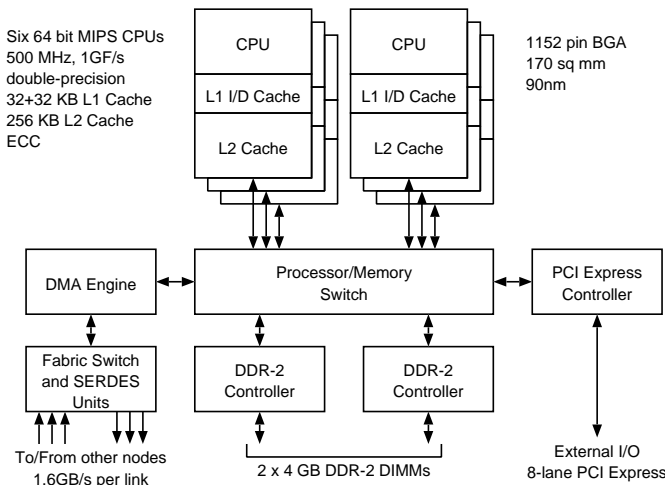


Fig. 1. SiCortex Node [2]

### B. Interconnect Fabric

A key point discussed in class and further explored in the lab was how SiCortex “tuned” the design of the machine. The relatively slow processor speed (to reduce power consumption) is complemented by novel hardware and interconnect implementations that together form a balanced platform in terms of performance, power consumption, and floor space. The interconnect consists of “links” that provide connections between nodes, the fabric “switch” for routing traffic within the node, and the “DMA Engine” that provides the software interface to the interconnect. The fabric links provide multiple 2 gigabyte/second connections between nodes without cables or switches. Elastic buffers are not required since the entire machine is frequency locked. The interconnect topology is a Kautz graph, which provides low network diameter and fault tolerance. Moreover, application software directly initiates messaging, bypassing the operating system. Details of the fabric switch implementation, packet format and primitives, virtual memory integration, and OS protection are described in [3].

The interconnect topology is a Kautz graph. Kautz graphs are interesting because they accommodate the largest known number of nodes for a given diameter (number of hops) using nodes of fixed degree (number of inputs/outputs). Kautz graphs are also naturally redundant for routing purposes. For example, the degree 3 graph shown in Figure 2 illustrates that between any pair of nodes there are three distinct routes that do not share any intermediate links or switches. Compared to other common topologies such as 2-D and 3-D tori the Kautz graph compares favorably in terms of diameter and bisection width for a given node count [1]. For example, to connect 972 nodes a Kautz graph of degree 3 requires a diameter of 6. A 3-D torus of degree 6 and requires a diameter of 15 while a 2-D torus of degree 4 requires a diameter of 32.

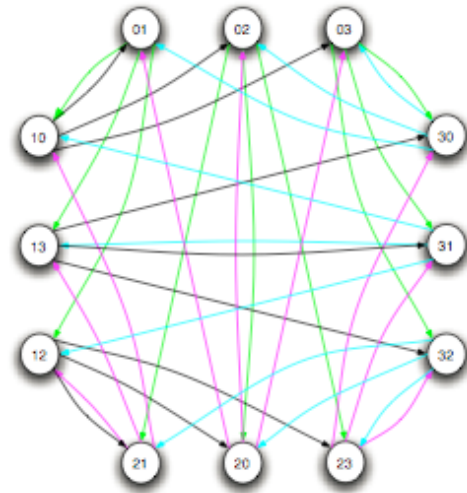


Fig. 2. 12-node Kautz Graph [3]

## IV. OPENFOAM

The OpenFOAM (Open Field Operation And Manipulation) CFD toolbox is an open-source computational fluid dynamics (CFD) toolkit developed by OpenCFD Ltd.. It consists of a set of solvers, utilities, and libraries that have been used to simulate anything ranging from fluid flows to solid dynamics, and even includes pricing for financial options [4].

The toolkit is mainly written in C++, allowing for an object-oriented approach to modeling cases. Each variable in the simulation, such as velocity or density, can become a class of its own. This provides an easier way to visualize the problem at hand and to set up the equations.

OpenFOAM cases use a three-dimensional mesh that can be refined to whatever level of granularity the user pleases. The solver can also operate on two-dimensional meshes if the user specifies an “empty” third dimension. These meshes can then be decomposed to run across any number of processors in balanced and unbalanced distributions.

### A. Compiling the Application

OpenFOAM is a very large and comprehensive toolkit. Furthermore, it has a large number of files in the source

due to its modular nature. While OpenCFD does provide x86 binaries for Linux, these binaries do not run on the MIPS64 architecture, so the application had to be compiled from source.

The first attempt at compiling OpenFOAM on one of the nodes ended in failure. The build process for the application does not use GNU make, but instead uses a homegrown make utility that then calls GNU make, along with a slew of other programs. The problem with this is that the source distribution includes x86 utilities such as mkdep in the build tree, but doesn't include a way to rebuild these utilities on other platforms. After finally rebuilding these utilities and increasing the maximum number of files that were allowed to be open by the user, the make process was able to start.

Initial builds of the toolkit ran for about 17 hours before erroring out. The problem was tracked down to a thermo-physical combustion library that would cause the compiler to segmentation fault.

Fortunately, SiCortex was able to contact Remik Ziemlinski, a software engineer for NOAA who happened to be working on using OpenFOAM on the SiCortex. Remik graciously agreed to provide MIPS64 binaries for the SiCortex. His solution was to use the cross-compiler on the SSP (an x86 machine), which both sped up the build process and allowed for the use of the x86 versions of flex and mkdep that came with the source distribution.

### B. Scaling Tests

Once OpenFOAM was compiled, the next step of preparation for the Cluster Challenge was to start running test cases and doing some scaling tests for the application. OpenFOAM solvers track two different times for the cases: execution time or CPU time, and clock time or total time – the execution time plus the time spent waiting on communication. As the number of cells per processor becomes smaller (number of CPUs goes up), there comes a point at which the problem becomes communication bound rather than CPU bound, and the overall runtime for the problem actually increases.

The first scaling test was done using the “*sonicFoam*” solver. SonicFoam is a “transient solver for trans-sonic/supersonic, laminar flow of a compressible gas” [5]. The problem was decomposed using a mesh of 59,600 cells with equal weights for each processor. Starting with one processor, the simulation was run to completion three times. The average clock and execution times of the three runs was calculated and recorded. Then the number of processors used was increased until 169 processors were used. At this point, the clock time was almost four times greater than the execution time, and the two times were growing farther apart.

In order to determine the best ratio of number of cells to number of processors, the execution time was used. It was found that the execution time peaked at about 500 cells per processor – a much lower number than the 10,000 cells per processor that was recommended on the OpenFOAM forums. The total wallclock time peaked a little earlier at about 1000 cells per core, but this could be due to an outlying datapoint

with a runtime of about 100 seconds longer than the other two runs. Figure 3 shows a graph of both the clock and execution times versus the number of cells per processor.

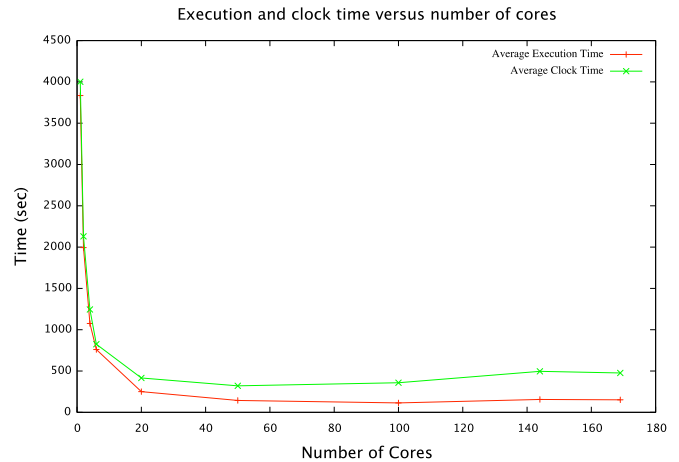


Fig. 3. Scaling tests for sonicFoam solver

A second test was performed using the “*oodles*” incompressible LES solver. This test consisted of approximately 13,800 cells and used a varying number of processors between 1 and 500. These runs saw an ideal window at about 81 CPUs, which yields 170 cells per core.

The other relationship observed during these tests was the ratio of the execution time to the overall clock time. As the number of cores increases, the ratio of execution time versus clock time diminishes. During the single core tests, this ratio was between 0.9 and 0.95. At the high end of both tests, the ratio was approximately 0.3. This meant almost 70% of the time was being spent waiting on communication and I/O than performing calculations.

### C. Challenge Data Sets

There were two OpenFOAM datasets for the 2008 Cluster Challenge. The smallest of the two data sets was a “*simpleFoam*” case, consisting of 195,000 cells. SimpleFoam is a “steady-state solver for incompressible, turbulent flow of non-Newtonian fluids” [5]. Due to the smaller cell count in this problem, the data set was more CPU bound for the Purdue machine. The data set completed in about six (6) hours without any errors or problems.

The second data set was a very large “*dieselFoam*” case made up of more than 4 million cells. The tarball for this problem was over 500 MB in size, and actually caused gigabytes of swapping to occur on the poor laptop being used to decompose the problem mesh. Once the problem was decomposed and enough cores were available for running the problem, the catch was spotted in the data set: while the problem only ran from Time = 0 to Time = 1, and the TimeStep was set to  $1 \times 10^3$ , the *controlDict* file allowed for the readjustment of the time step if the problem size required it. Unfortunately for all of the teams in the Cluster Challenge, this meant the TimeStep was adjusted to  $1.33 \times 10^{-7}$ , meaning

the problem would take most of the 48 hours available to complete (on our SiCortex). Since this problem wasn't started until about 12 hours into the competition, there was no chance of the Purdue team finishing this dataset.

## V. THE WAVE PROPAGATION PROGRAM

The Wave Propagation Program, or WPP, is maintained by Lawrence Livermore National Laboratory. The program simulates the propagation of different types of waves across a grid on a parallel computer. WPP includes significant functionality for simulating 3d seismic events and has been used to simulate popular earthquakes [6].

After learning that this application was going to be one of the many programs used to evaluate our cluster at the Cluster Challenge, effort was put into gathering and compiling all the requisite libraries and then working out various bugs with Sicortex and Pathscale to produce a correct binary. The main challenge in getting WPP compiled was the use of deprecated declaration of strings that the Pathscale compiled was not able to handle correctly. Once a working binary was available, testing begun on the example input sets provided in the WPP source distribution to see how well the program would scale on our Sicortex.

### A. Scaling Tests

The first scaling test was completed using a simple input set modeling the movement of a wave in a half sphere. The grid was set to a cube with x, y and z lengths of 10000, 10000 and 5000 [7], with a block size of 62.5 and three listening stations. Running this input set using WPP produced a scaling graph that can be found in Figure 4.

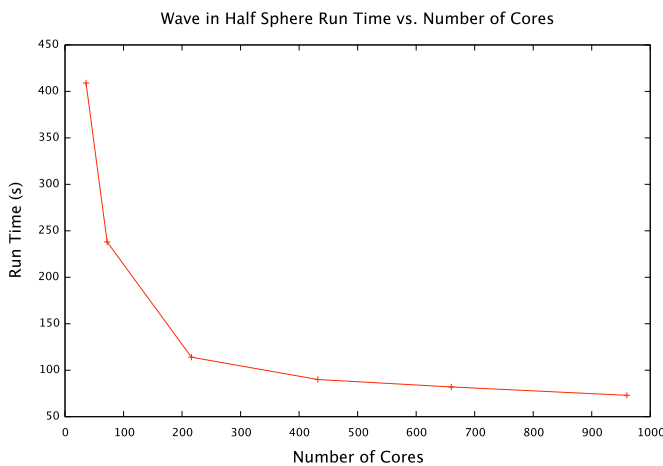


Fig. 4. Wave in half sphere scaling results

Given more processing cores continued to lower the runtime until the maximum number of cores was reached on the machine. Based upon the slope of the lines connecting the data points in the graph, improvement in runtime verses the number of cores used starts to dropped off somewhere near 200 cores.

The second data set tested before the competition was the same problem except with a larger block size (500) and three "refined" regions that increase the resolution [8]. The graph of the runtimes in Figure 5 shows more interesting scaling performance than for the previous problem. Between 72 and 200 cores, the run time drops very quickly with additional cores. Doubling the number of cores nearly doubles the performance of the application on this particular data set. Then performance quickly levels off before becoming worse after 400 cores.

Refinement statements appeared to have interesting effects on the run time of data sets, but scaling still started to suffer after 200 cores.

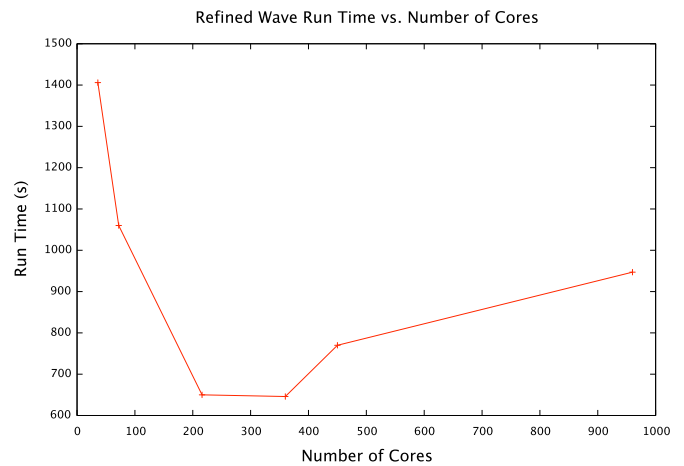


Fig. 5. Refined wave in half sphere scaling results

### B. Challenge Data Sets

The data sets from the Cluster Challenge simulated a large grid with one source and multiple listening points. The cube was  $30000 \times 30000 \times 17500$  with varying block sizes: 35, 40, 45 and 50. At the challenge, an insufficient number of cores were used for the data sets and no completed output data was collected. After returning from the Cluster Challenge, a series of runs were performed on the data set with a block size of 35.

Using the Purdue SiCortex, it was determined that at least 450 cores were required for the program to run to completion. Collecting the run time of five runs using 450, 660, and 960 cores were completed and averaged, and are shown in Figure 6. It can be seen that the data set scales well upto the maximum number of cores in the SiCortex taken to the Cluster Challenge.

At the competition, each SiCortex node had access to 4GB of memory and the runs were given approximately 48 hours to complete using 25 nodes, or 150 cores. Runs using the Purdue SiCortex showed the data set consumed 45GB of memory, give or take several megabytes depending on the number of nodes used. The possible bottleneck at the competition was the number of cores thrown at the problem. The team attempted to balance the queue so that every application had a chance at

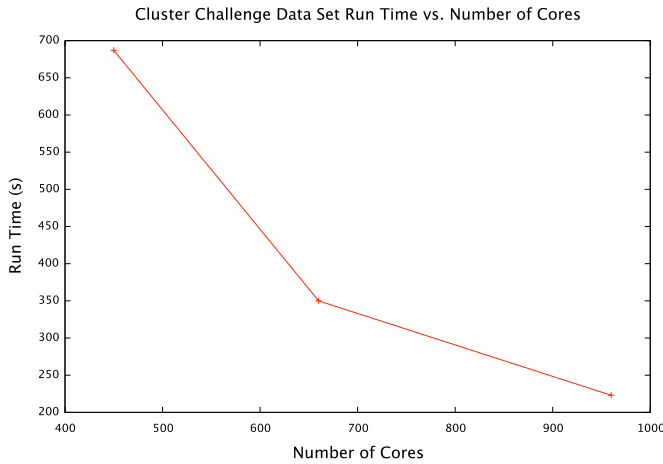


Fig. 6. Cluster Challenge data set scaling results

running, which is why a relatively low number of cores were given to WPP runs.

## VI. CONCLUSION

This year’s cluster challenge combined many different applications, each of which stressed different parts of clustered computers. Both OpenFOAM and WPP provided unique challenges and interesting scaling effects when used on a cluster with a high count of cores.

The Purdue team this year chose to attempt more high throughput queuing, giving each application a slice of the machine to run for the competition. With the available data at the start of the competition, it appeared this was a reasonable way of scheduling runs. However, the data sets given at the competition did not lend themselves to the plan. In the future, the team will probably look more closely at which applications will always scale well on the cluster and prioritize those applications over others that have poor or unknown scaling properties given a high core count.

Thinking in terms of many cores was a change in strategy from last year’s competition, in which the Purdue team used all commodity hardware. It was definitely a curve ball, but the very high integration of components in the SiCortex provided other advantages. The vendor provided great support heading into the competition and was always available to provide help when they could. The power system of the SiCortex lends itself well to balancing power draw from two power strips. Other teams spent a lot of time making load adjustments and had to closely watch their power balancing across circuits, but the SiCortex allowed for a “set and forget” method of power monitoring at the competition.

In fact, the SiCortex was well enough integrated that the Purdue team was able to push the machine to become the first machine at the competition to achieve 1TFlop using the Linpack benchmark and provided great enough power efficiency to win the “Cluster Challenge 2008 Most Power Efficient” award.

However, bringing this technology did come at a cost. A lot of work had to be done before applications could even run on the machine. In the case of the POY application, a software dependency on the OCaml compiler prevented the team from even building the application for MIPS.

The slower clock speeds of the MIPS cores affected overall runtime for CPU-intensive code that didn’t necessarily benefit from a low-latency interconnect. On the other hand, having more processors meant more jobs could be run simultaneously with room for the parallel applications to use many cores.

With more experience in evaluating scientific applications and a crash-course using high core count computing, this years Cluster Challenge will serve as a great stepping stone for future competition efforts.

## VII. ACKNOWLEDGEMENTS

The authors would like to thank SiCortex for their amazing support of the team in preparation for and at the SC08 Cluster Challenge. We would also like to thank Remik Ziemiński for his help in getting OpenFOAM running on the system.

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

The rest of the Purdue Cluster Challenge team: David King, Paul Willmann, and Ryan Weinschenk.

Finally, we would like to thank the entire Cluster Challenge committee for allowing us the opportunity to return to the Cluster Challenge.

## REFERENCES

- [1] Jud Leonard. The kautz digraph as a cluster interconnect. Technical report, SiCortex, Inc., 2007.
- [2] Nitin Godiwala, Jud Leonard, and Matthew Reilly. A network fabric for scalable multiprocessor systems. In *HOTI '08: Proceedings of the 2008 16th IEEE Symposium on High Performance Interconnects*, pages 137–144, Washington, DC, USA, 2008. IEEE Computer Society.
- [3] Lawrence C. Stewart and David Gingold. A new generation of cluster interconnect. Technical report, SiCortex, Inc., April 2008.
- [4] OpenFOAM Website [<http://www.opencfd.co.uk/openfoam/>]
- [5] OpenFOAM Manual [<http://www.opencfd.co.uk/openfoam/doc/user.html>]
- [6] Daniel Appel, et al. User’s guide to the Wave Propagation Program version 1.1. Lawrence Livermore National Laboratory, 2007.
- [7] Lamb’s Problem, Sample Data Set Layer.in from the WPP v1.1 Source Repository Examples. Lawrence Livermore National Laboratory, 2007.
- [8] Lamb’s Problem Refined, Sample Data Set Layer.in from the WPP v1.1 Source Repository Examples. Lawrence Livermore National Laboratory, 2007.