Benchmarking a Large-Scale Heterogeneous Cluster

Daniel Loreto, Erik Nordlander, and Adam Oliner
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139-4307 USA
e-mail: {loreto, erikn, oliner}@mit.edu

Abstract

Many of the world's fastest supercomputing systems are clusters of commodity hardware connected via a high-speed network. Typically, these clusters consist of homogeneous nodes and a uniform network. The Alliance for Computational Earth Science (ACES) at the Massachusetts Institute of Technology owns a set of four clusters, consisting of hundreds of CPUs. Unlike most supercomputing clusters, the ACES Grid is heterogeneous in nearly every respect: architecture, network connection, memory, processor speed, etc. This heterogeneity provides numerous technical challenges in addition to the ones that are normally involved with running and tuning a benchmark on a large-scale system. It was our experience, however, that these technical challenges were not the primary impediments to benchmarking. This paper recounts those experiences and advises future benchmarking efforts. Most importantly, we perform a number of smaller benchmark experiments, and use them to project the performance of the entire cluster.

1 Introduction

Large-scale clusters of commodity machines are increasingly being used for scientific computing. The MIT ACES Grid [1] is one such cluster, consisting of hundreds of machines across multiple buildings. The High Performance Linpack (HPL) benchmark [5] is the standard code for measuring the performance of high performance computers, and is used for ranking on the Top500 Supercomputer List [6]. The stated goal of this project was to benchmark the ACES Grid, with hopes of getting MIT listed on the Top500.

Toward that end, HPL was compiled for multiple platforms and run across both different architectures and different buildings. Extensive testing was done to determine the optimal combination parameters for the benchmarks, and we evaluated multiple implementations of MPI and linear algebra libraries. In the end, we were unable to actually benchmark the entire cluster. Instead, numerous runs were performed with the aim of using them to make performance projections.

The technical challenges we anticipated were not ultimately the same challenges that made benchmarking the entire cluster impossible. The tremendous heterogeneity, inter-building network topology, algorithmic limitations, and enormous search space for parameter selection were all tractable. Instead, issues with cluster reliability and consistency were the central impediments to success.

This paper describes the cluster, the benchmark, the benchmarking process, the results, and what we learned. Specifically, Section 2 describes the ACES Grid in more detail. In Section 3, the HPL benchmark is covered, including the process of parameter selection and some example outputs and errors. Results are presented in Section 4, along with a projected performance number for the full cluster. Finally, Section 5 provides five rules for benchmarking a large-scale cluster and enumerates the contributions of this paper.

2 MIT ACES Grid

The Alliance for Computational Earth Science (ACES) at the Massachusetts Institute of Technology controls a set of four clusters, collectively called the ACES Grid. Although the ACES Grid website [1] indicates that there are 566 CPUs with 829 GB of memory, and a theoretical peak performance of 2.706 TFlops, the actual hardware that was available to us differed significantly from that. The cluster we attempted to benchmark consisted of ~ 334 functional CPUs with 339 GB of memory, and a theoretical peak of around 1.464 TFlops.

2.1 Discovery Script

This disparity led to the development of a hardware discovery script. In order to fully explore the capabilities of the machines available for benchmarking, we developed a script that, given a master list of machines, can verify and report certain important pieces of information. We started with a list of machines that were supposed to be available. The script attempts to login to each of these machines, collects hardware information and then compiles a report at the end.

This script was partially necessary because of the heterogeneity of hardware available in the cluster and partially because machines were often down and we needed to identify them before attempting to run HPL. Furthermore, some of the stated hardware attributes of the cluster incorrect, so we needed to know what hardware the cluster machines actually provided.

The script allowed us to get an accurate picture of how many physical processors were available. Some Pentium 4s had hyperthreading technology enabled which caused earlier counts to be erroneous. It also measured the total memory available and the minimum physical memory on any node in the cluster, which is an important variable because HPL divides the problem evenly among nodes. The problem size would be limited by the machine with the least available memory. Finally, the current load was checked on each machine because it was important to verify that jobs were not being run on the machines (circumventing the scheduler) before attempting the benchmark.

The script was developed in Python and probed the Linux devices on each machine.

2.2 Heterogeneity

Aside from the obvious challenges of optimizing performance, the ACES Grid's heterogeneity provided additional challenges:

Platform Heterogeneity The system includes machines from numerous different CPU architectures. There are 140 dual-processor Xeon nodes, 9 single-processor Xeon nodes, 45 Pentium 4 nodes, 3 Itanium2 compute nodes, and 1 Itanium2 Altix compute node. By CPU, that is 289 Xeons, 45 Pentium4s, and 19 Itanium2s. These numbers are constantly in flux, but stand at the time of this writing. In order to optimize performance, it is vital to compile different HPL binaries for each architecture. Furthermore, they may run different platform implementations of MPI.

Clock Speed Heterogeneity There are also multiple clock speeds, even within architectures. The Xeons range from 2.4-2.8 GHz, the Pentium 4s from 2.26-2.8 GHz, and the Itanium2s are running at a distant 1 GHz. This raises interesting performance questions: Will the slowest machines govern the performance of HPL? Can the work be distributed such that CPU bottlenecks are negated?

Memory Heterogeneity The system contains 339 GB of memory, but this is not evenly distributed across nodes. The disparity is rather large; some of the Pentium 4s have only 1 GB of RAM, while some Itanium2 CPUs have 8 GB of RAM, each. Without fine-tuned load balancing, this may severely limit the size of the problem able to be tackled by the system. Because the performance of HPL tends to increase with the size of the problem, this may prove critical.

Network Heterogeneity Network performance, though not as central to the Linpack benchmark, is nevertheless crucial. Unfortunately, the speeds of the interconnects in the ACES Grid vary almost as dramatically as every other aspect. Although most of the nodes are connected via Gigabit Ethernet, some have faster Myrinet interconnects and others have only 100 Mb/s Ethernet. Initial tests indicate that inclusion of nodes on this slower network degrades overall performance by an order of magnitude. Additionally, the clusters are located in different buildings around campus, connected by 10 Gigabit links. This raise additional questions: Will those nodes on a slow network need to be excluded from the final benchmark? Again, is there some way to hand-tune the load balancing to compensate for this heterogeneity?

2.3 Reliability and Consistency

The cluster was constantly in a state of flux regarding the status, location, and behavior of the nodes. Nodes were down frequently. At one point, the failure of a cooling mechanism resulted in the destruction of important routing hardware. After that disaster, there were configuration and reliability issues with the new hardware the persisted through the termination of this project. Machines, though named according to their original building, were prone to move without warning or notification. This made it difficult to test for the effects of physical distance.

Although the nodes were mostly under schedulers, they were not all under a scheduler and certainly not all under the same scheduler. Benchmarking requires control over the hardware, control which should come, in part, from the schedulers; instead, users would sometimes run code on machines by circumventing the schedulers, and it was impossible to regulate access to those machines not under schedulers. Because the cluster was constantly being used, and the routing and network hardware was shared among many machines, it was not possible to control for the effects of other users' code without clearing off the entire cluster.

3 High Performance Linpack

In order to be ranked on the Top500 [6], a supercomputer must solve a dense system of linear equations as prescribed by the Linpack benchmark. Linpack provides, among other metrics, a performance in billions of floating point operations per second (GFlops) that is used for the ranking. The original Linpack paper [3] provides the performance of many computers, while the recent survey [4] gives a more complete history of the benchmark and its current incarnations.

One popular implementation of Linpack is the High Performance Linpack (HPL) benchmark [5]. Specifically, HPL solves an order N system of the form $Ax = b$ by determining the LU factorization: $Ab = LUy$. The solution (x) is subsequently computed by $Ux = y$. The matrix is divided into $NB \times NB$ blocks that are cyclically dealt onto a $P \times Q$ grid with a block-cyclic layout. Further details about the algorithm can be found at the HPL website.

3.1 Parameters

HPL itself has a plethora of settings to be adjusted for individual systems. These are set inside an input file to the HPL executable, called `HPL.dat`. These options include broadcast patterns, process mappings, and lookahead depth. Although a number of these parameters tend to have consistently optimal settings, many needed be tested for this particular cluster setup; this combinatorial tuning process was very time consuming. One important setting is the problem size N (where the algorithm solves a dense, random $N \times N$ system of equations), which is limited by the memory per node. Some crucial parameters and sample values are outlined in Table 1. Beyond these parameters, it is permissible to modify the HPL algorithm to take advantage of system-specific characteristics.

Parameter	**Meaning**	**Value**
N	Solves an $N \times N$ linear system	65,536
NB	Data distributed in $NB \times NB$ blocks	96
P, Q	Processors arranged as $P \times Q$ grid	4×4
BCAST	Type of broadcast (panel distribution pattern)	3
DEPTH	Lookahead depth	1

Table 1. Small subset of the HPL parameters and example values. Determining the best combination of parameters requires experimentation, but there are certain guidelines and heuristics to guide this search. For example, $P \times Q$ should be as nearly square as possible, and NB is typically optimal within $[32, 256]$.

The problem size is limited by the amount of memory on the machine with the least physical RAM. To determine the maximum problem size, we used a calculation suggested elsewhere [2], where N is the problem size, and M is the total amount of available memory in bits (calculated by multiplying the bits of memory in the smallest machine by the number of machines):

$$N = 0.8\sqrt{\frac{M}{8}}$$

As a rule of thumb, 80% of the total available memory may be used by HPL, while 20% is left for the operating system and other user processes. Each matrix entry is one byte. This calculation is important, because performance improves as the problem size increases. It is, therefore, desirable to run with as large an N as time and hardware permit.

3.2 Output

After computing the solution to the linear system, HPL checks its work by computing three residuals that must all fall within a threshold for the results to be considered valid. An example output follows, in which a problem of size $N = 8,192$, with $NB = 96$, was computed on a 4×4 processor grid. The run took 12.54 seconds and gave a performance measure of 29.25 GFlops. That performance number is ultimately the value the HPL is meant to measure. All the residual tests were passed:

```
=====================================================================
T/V                    N    NB     P     Q        Time          Gflops
---------------------------------------------------------------------
WR13R2C8            8192    96     4     4       12.54       2.925e+01
---------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) = 0.0081865 ...... PASSED
```

```
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) = 0.0034536 ...... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) = 0.0005700 ...... PASSED
====================================================================
```

3.3 Errors

As with many MPI programs, HPL does not fail gracefully, nor informatively:

```
p8_16259:  p4_error: net_recv read:  probable EOF on socket: 1
p9_16285:  p4_error: net_recv read:  probable EOF on socket: 1
p7_18662:  p4_error: net_recv read:  probable EOF on socket: 1
...
bm_list_24227: (3656.323591) wakeup_slave: unable to interrupt slave
     0 pid 24224
bm_list_24227: (3656.323703) wakeup_slave: unable to interrupt slave
     0 pid 24224
...
Broken pipe
Broken pipe
```

These errors would often happen nondeterministically, though more frequently with larger problem sizes or longer runs. The broken pipe errors became symbolic of the reliability issues we experienced with the cluster.

3.4 Compilation

There are innumerable options and parameters to consider when compiling HPL, and which may influence the performance of the benchmark. First, there is a choice of MPI implementation. For example, the VMI implementation would allow us to use the Myrinet interconnects, while the standard Intel MPICH implementation does not, but has inherent support for explicit load balancing. Second, HPL can be compiled with a choice of linear algebra library package; options include the Goto libraries, Intel's Math Kernel Libraries (MKL), and Atlas. The libraries often come in both single-threaded and multi-threaded flavors.

Although it would have permitted the use of Myrinet, VMI was not easily capable of supporting cross-platform execution. In particular, it did not have support for machinefiles that permit the specification of separate binaries for each machine entry. This ruled out VMI as an option. Furthermore, Myrinet was not working for much of the project, so the point was moot. We used Intel MPICH as the MPI implementation. After a series of experimental runs using different algebra packages, it was clear that the Goto libraries provided superior performance. HPL was compiled for both the single and multi-threaded Goto variants, to be used for larger scale testing. The compiler flags used for each platform are included in Table 2.

4 Results

A full cluster run was ultimately not possible, so we decided to provide an estimate for the computational power of the cluster should conditions permit it to be properly benchmarked. In order to make this projection, a large number of smaller runs were performed in order to estimate scaling at larger sizes.

Prior to a cooling failure disaster in which hardware was destroyed, we were able to perform some larger runs. In particular, one run was able to achieve 442 GFlops on 182 CPUs with a problem size of $N = 100,000$. This run used exclusively Xeons and the single-threaded Goto library. Considering the number of CPUs, this performance number is actually rather low; it is only around 50% of what we expect. Those machines, however,

Architecture	Algebra Package	Flags
Xeon	GOTOP	`-O2 -xN -tpp7 -unroll -pthread`
Pentium 4	GOTO	`-O2 -xN -tpp7 -axW -unroll`
Itanium2	GOTOP	`-O2 -unroll -lmpi -lpthread`

Table 2. Compiler flags used for compilation of HPL, listed by architecture. The flags listed are for the Xeons and Itaniums were for the multi-threaded algebra packages, and the Pentium flags are for the single-threaded package.

should be able to handle a problem as large as $N = 500,000$. That sized problem should improve performance to where we anticipated it. Since the destruction of the hardware, unfortunately, the runs have been limited to very small sizes and numbers of processors.

4.1 Benchmarking the Cluster

The majority of the ACES Grid cluster is made up of dual processor Xeons and single processor Pentium 4 systems. The total breakdown is described in the ACES Grid section (Section 2).

Because large scale tests were not often possible, we did several smaller tests varying the number of processors, problem size, and mix of systems. This allows us to measure how the performance of the cluster scales under various conditions and approximate the total performance of the cluster.

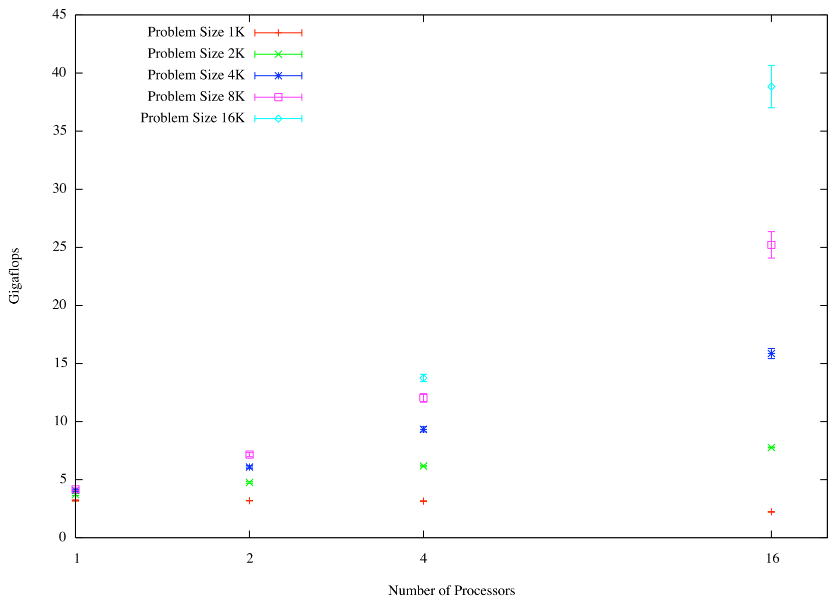

Figure 1. Various HPL tests on Xeon systems. The number of processors and problem size (N) were varied and the performance in GFlops was measured. As expected, HPL scales well with both parameters. Error bars indicate the standard error obtained by running the experiment five times.

HPL runs were done on groups of machines with 1, 2, 4, and 16 processors. The problem size was varied from $N = 1024$ to 16384. Larger problem sizes were attempted but produced an array of broken pipe errors. Our calculations indicated that larger problem sizes should have worked with given the Xeon nodes available and their respective physical memory sizes, but for unknown reasons HPL was unable to produce a benchmark. We have reason to believe this is the result of problems with the cluster, because significantly larger runs of HPL were completed with the same binaries and settings. Specifically, we believe this may be the result of lingering hardware problems since attempting to isolate problem nodes proved to be a very difficult task. Nodes that would fail in one run might perform without any problems in similar run at a later time.

The runs above were performed at least five times each in order to account for variance between runs. For the most part, the disparity was insignificant, often less than 30 MFlops. This is meaningless given that a single processor is often 100 times faster than this. For the largest runs on the highest number of processors, however, we observed errors of around 1 GFlops (about 4% of total performance) which was taken into account in our final predictions of the theoretical cluster performance.

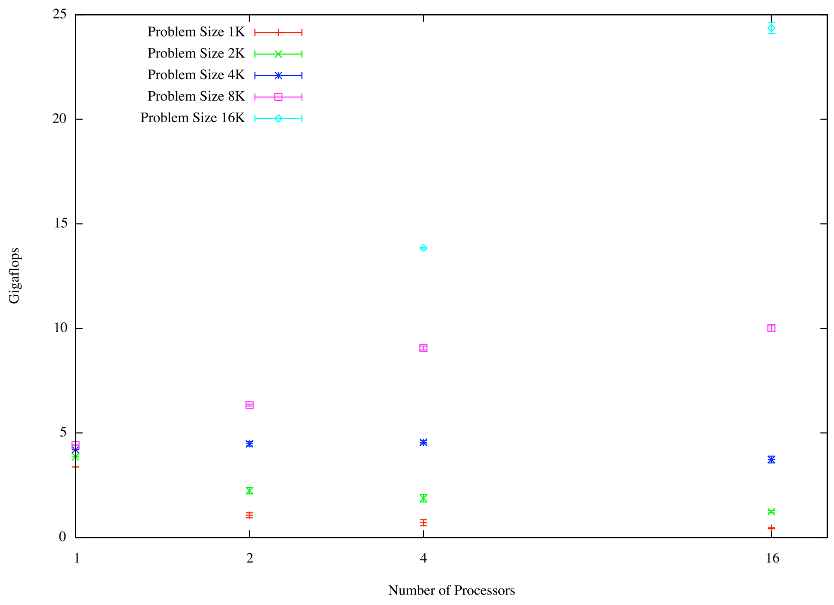

Figure 2. Various runs of HPL on Pentium 4 systems. The trends matched the Xeon runs, but performance was uniformly decreased. Error bars indicate the standard error obtained by running the experiment five times.

Another interesting result is that, for the smallest test size, performance decreased when using 16 processors. This demonstrates that the time to set up and distribute the problem can dominate over the time needed to perform the actual computation. Indeed, this setup time is one reason that larger problem sizes are so profitable from a performance standpoint; the setup time is amortized over the longer computation.

The Pentium 4 systems followed the same trend as the Xeons except for a reduction in performance across the board. This was expected because of the slightly different archetecture and cache size available on the Pentium 4s. Some Pentium 4s also had a reduced processor clock speed. With the Pentium 4s there is a drop in performance from 1 to 2 processors with the smaller problem sizes for similar reasons as the drop in performance for 16 processors on the Xeon systems. These problems are not large enough to exploit the parallelism with larger number of processors and thus the running time is dominated by distributing the problem.

To determine the effect of mixing architectures, we ran HPL tests with an equal mix of Pentium 4s and Xeon

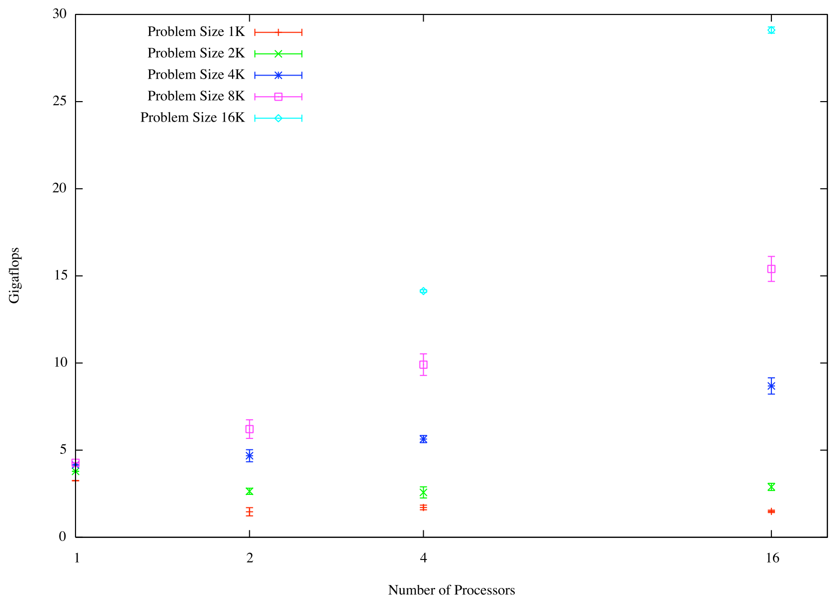

Figure 3. Runs of HPL using mixed Xeon and Pentium 4 architectures. For a given run, half of the processors were of each architecture. Error bars indicate the standard error obtained by running the experiment five times.

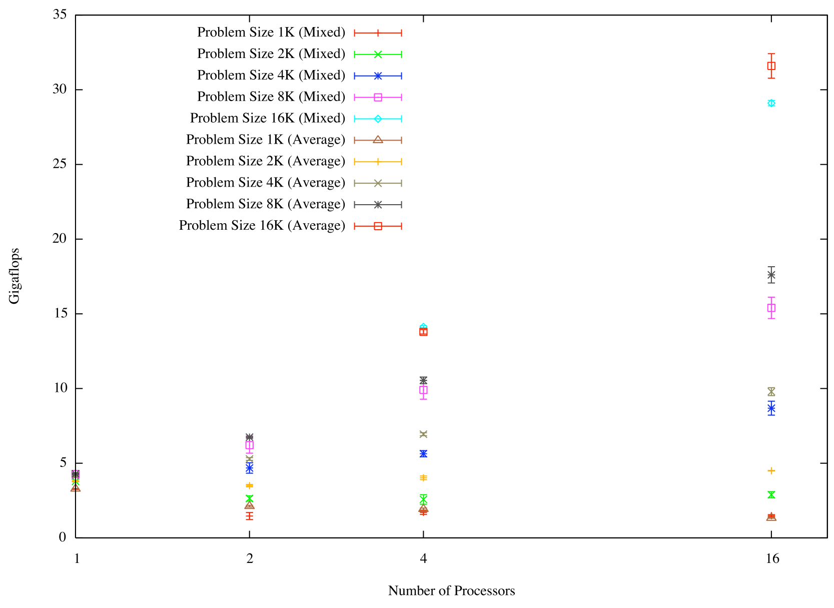

Figure 4. Comparison between runs using mixed Xeon and Pentium 4 architectures, and the predicted performance obtained by averaging the pure Xeon and the pure Pentium 4 runs.

nodes. This does not reflect the composition of the total cluster, but we can learn quite a bit from this equal ratio. In the tests we ran, we noticed that the performance numbers were close to an average of those of the pure Xeons and pure Pentium 4s HPL tests. The numbers are slightly lower than a perfect average which may be due to other performance considerations when performing a multi-platform run. This allows us to perform simple weighted average to estimate the effect of the presence of the Pentium 4s in the total cluster estimate.

4.2 Full Cluster Estimate

We used the results from the above mentioned experiments to predict the performance of the whole cluster.

Our mixed experiment results indicated that it was a reasonable estimate to perform a weighted average of the P4 and Xeon results according to the number of processors of each type. For this estimate we assumed that we had a total of 334 processors available, 289 of which were Xeons and 45 of them P4s.

After calculating the weighted average, we used a least square fitting algorithm to find the optimal line passing through the data points we had. Notice that both the problem size and the number of processors affect the performance measure obtained by the HPL benchmark. To account for these factors, each of our projected lines passes through data points were the ratio between problem size and number of processors is constant. That is, our projection assumes that as we increase the number of nodes, we increase the problem size appropriately to ensure that each node still has the same load.

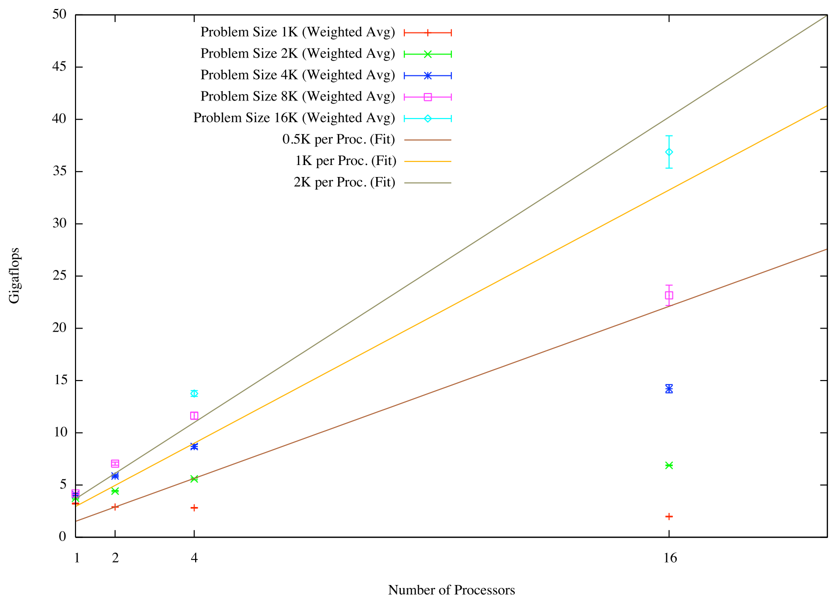

Figure 5. Projected performance of the whole cluster based on the results obtained from the previous experiments. Each projection line assumes that the ratio between problem size and number of nodes is mantained constant.

Because we want to increase performance as much as possible, and a better performance is usually obtained with a bigger problem size, we base our estimate on the projection with the highest problem-size/node ratio. The resulting equation for that projection is $f(n) = 2.43613 \cdot n + 1.24252$. Which yields a performance estimate of 814.19 GFlops for 334 processors, and 1.380 TFlops had we had all the 566 processors described on the ACES Grid webpage. Note that our projection is fairly accurate; for $n = 182$ it yields 444.618 GFlops which is very close to the 442 GFlops we empirically obtained using 182 CPUs.

Given our estimates, we believe that, had we had access to the 566 nodes, we could have easily helped MIT regain a spot on the Top500 list. Using only the 334 CPUs that are currently available, our estimate implies an utilization of about 55% and falls a bit short of the desired 1 TFlop necessary to get to the Top 500 list. It is possible, however, that with a bigger problem size and some optimizations we could have pushed the cluster to 68% utilization and over 1 TFlop.

5 Conclusions and Contributions

This paper has described efforts to benchmark the MIT ACES Grid. HPL was compiled for and run across multiple platforms, including Xeons, Pentium 4s, and Itaniums. A hardware discovery script was developed to dynamically search and characterize the available hardware. Ultimately, the cluster was able to achieve 442 GFlops on 182 CPUs using a problem size of $N = 100,000$. That run was performed on Xeons using the single-threaded Goto library across two buildings. Extensive experiments and projection calculations suggest that scaling to the 305 available CPUs would result in 0.547 TFlops; scaling to the hypothetical 566 CPUs would yield 0.966 TFlops.

This project makes the following contributions:

- Introduces the ACES Grid and explains the HPL benchmark.
- Describes the compilation and running of HPL across architectures.
- Presents results from numerous benchmarking experiments.
- Projects the performance of the complete cluster.
- Proposes five guidelines for future benchmarking efforts to consider.

5.1 The Five Rules

In light of our experiences, we suggest five rules or guidelines for people proposing to benchmark a heavily-used large-scale cluster like the ACES Grid.

1. **Know thy cluster.** It is important to have a complete and accurate picture of the hardware in the cluster and how it is interconnected. Do not trust static records; clusters change constantly. We suggest a hardware discovery script such as the one we used.
2. **Get root.** Installing software that is necessary for the benchmarks should not need to go through a chain of command. Insist on having the privileges necessary to do the job.
3. **Get keys.** Nodes sometimes require physical intervention, such as if they hang on restart. Gain access to the machines directly.
4. **Be self-important.** Benchmarking needs to be a top priority, or it will not get done. Users have work to get done on the cluster, but do not be afraid to schedule time for benchmarking and to kick off users when needed.
5. **Understand politics.** Benchmarking is not popular. Be aware that users may be hostile toward the idea of the cluster being cleared so that someone can check how fast it is. Sell them on the idea, and assure them that it is to their benefit.

References

[1] ACESGrid. Acesgrid.org: About & news. URL: http://acesgrid.org/, 2005.

[2] BeowulfBenchmarking. Summer 2004 beowulf benchmarking project. URL: http://10.pins1.xdsl.nauticom.net/bvds/computing/Beowulf_Benchmark.htm, 2004.

[3] J. J. Dongarra. Performance of various computers using standard linear equations software in a Fortran environment. In W. J. Karplus, editor, *Multiprocessors and array processors: proceedings of the Third Conference on Multiprocessors and Array Processors: 14–16 January 1987, San Diego, California*, pages 15–32, San Diego, CA, USA, 1987. Society for Computer Simulation.

[4] J. J. Dongarra, P. Luszczek, and A. Petitet. The linpack benchmark: Past, present, and future. URL: http://www.cs.utk.edu/∼luszczek/articles/hplpaper.pdf, 2002.

[5] HPL. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. URL: http://www.netlib.org/benchmark/hpl/, 2005.

[6] Top500. Top500 supercomputer sites. URL: http://top500.org/, 2005.