Cluster Challenge 2008: Optimizing Cluster Configuration and Applications to Maximize Power Efficiency

Jupp Müller[1], Timo Schneider[2],
Jens Domke[1], Robin Geyer[1], Matthias Häsing[1], Torsten Hoefler[2],
Stefan Höhlig[1], Guido Juckeland[1], Andrew Lumsdaine[2], Matthias S. Müller[1]
and Wolfgang E. Nagel[1]

[1] Center for Information Services and High Performance Computing (ZIH)
TU Dresden
01062 Dresden, Germany
{jupp.mueller,jens.domke,robin.geyer,stefan.hoehlig,
guido.juckeland,matthias.mueller,wolfgang.nagel}@zih.tu-dresden.de,
[2] Open Systems Lab
Indiana University
Bloomington, IN 47405, USA
{timoschn,htor,lums}@osl.iu.edu

Abstract. The goal of the Cluster Challenge is to design, build and operate a compute cluster. Although it is an artificial environment for cluster computing, many of its key constraints on operation of cluster systems are important to real world scenarios: high energy efficiency, reliability and scalability. In this paper, we describe our approach to accomplish these goals. We present our original system design and illustrate changes to that system as well as to applications and system settings in order to achieve maximum performance within the given power and time limits. Finally we suggest how our conclusions can be used to improve current and future clusters.

About our team

The *ClusterMeister* team consisted of six undergraduate students, three of them from Technische Universität Dresden, namely Jupp Müller, Jens Domke and Robin Geyer. Two students from Indiana University, Valkyrie Savage and Chris Beckley and one student from Technische Universität Chemnitz, Timo Schneider, who was a visiting student at Indiana University at the time the challenge took place. The team was guided by two advisors, Torsten Hoefler from Indiana University and Guido Juckeland from TU Dresden. Our team was supported by two vendor partners, IBM and Myricom. Despite the fact that our team was international and some of the team members never met each other prior to the start of the challenge, we managed to outperform our competition at Cluster Challenge 2008.

1 Introduction

Getting the best performance is still the first priority in high performance computing. However, the increasing power consumption is often a limiting technical or financial factor nowadays. It gains more importance with growing power costs. The increasing costs of electrical energy cause growing concern and recently, the Green 500 list [1] was started to monitor the power efficiency of the largest supercomputers. However, extensive studies about techniques to minimize the power consumption and maximize the output of high performance systems are yet to be done. Figure 1 shows the average price for electrical energy according to the October 2008 Monthly Energy Review [2] of the US Energy Information Administration. The trend of increasing energy prices is expected to continue. A new competition, the Cluster Challenge, was started in 2007 in conjunction

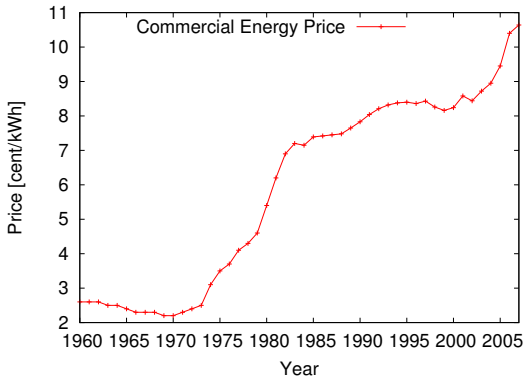

Fig. 1. Development of Electrical Energy Prices in the US

with the Supercomputing Conference to promote the research of those problems and to motivate students to join the high performance computing society.

The goal of the challenge is to form a team of six undergraduate students in order to design, build and operate a cluster built from commercially available components to run different scientific applications as well as the High Performance Computing Challenge (HPCC) Benchmark [3]. The challenge addresses energy efficient cluster computing by limiting the total power consumption of each team's cluster. Since power consumption and time are the only limiting factors, the teams have to optimize their systems purely for maximizing the performance within the given power budget. Thus, the main problem is to optimize the power efficiency of the overall system.

This paper has two main sections: the description of the computing platform used and the changes we applied to optimize its power efficiency as well as the the desciption of the applications used and their performance optimizations. While

some subsections in section 2 describe how known techniques have been applied to our cluster (e.g. hard disk suspends), others present the students' research on methods for additional power savings (e.g. power distribution).

2 Computing Platform

Each team of the Cluster Challenge worked with vendor partners who sponsored the hardware. Our sponsor IBM provided us with an System x iDataPlex dx360 cluster system consisting of 14 nodes while our second sponsor—Myricom—sponsored the latest Myrinet MX interconnects. Since we also had the chance of being supplied with InfiniBand ConnectX cards, we had to choose which interconnection network technology we wanted to use, as described in section 2.3. All nodes had the identical hardware configuration shown in Table 1.

The initial application optimization work was done on clusters at Zentrum für Informationsdienste und Hochleistungsrechne (ZIH) in Dresden. Since these clusters are running SuSE Linux Enterprise Server 10 SP2, the same Linux distribution was chosen for the competition cluster to keep the results reproducible and leverage the experience gained on the ZIH systems.

Table 1. IBM System x iDataPlex dx360 initial hardware configuration

Description	Count	Item
Processor	2	Intel Xeon L5420
Memory	4	4 GiB Green DIMMs
Network Card (Option 1)	1	Mellanox ConnectX IB (MT26418)
Network Card (Option 2)	1	Myrinet MX (10G-PCIE-8B-QP)
Hard Disk	1	SATA Western Digital 250 GB

2.1 Power Distribution

Motivation The Cluster Challenge rules specify the use of two wall plugs, each current metered and not allowed to exceed the maximum of 13 A. The obvious procedure is splitting up all the hardware into two parts M_1 and M_2 that draw close to 13 A each while not exceeding that limit. We will label the two currents coming out of both plugs $i_1(t)$ and $i_2(t)$ respectively with $i_1 = \hat{I}_1 \cos(\omega\, t + \varphi_1)$ and a similar definition for i_2. We further define I_1 being the root mean square (rms) current for i_1 and the same definition for I_2. When we considered taking this easy approach, neither the split-up itself nor the management of the two power partitions proved to be trivial.

We found out that it was not possible to find a configuration where the 14 nodes and the Myrinet switch drew equal currents from each wall plug, mainly

because every two nodes shared one power supply which made it impossible to use 7 nodes on each wall plug.

Another important aspect was the ability to appropriately distribute computing jobs among those two power partitions. To illustrate the difficulties, imagine M_1 with 6 nodes and M_2 with 8 nodes. Job A runs on 3 nodes of M_1, job B on 4 nodes of M_2 and a cross-partition job C on 3 nodes of M_1 and 1 node of M_2. The currents result in $I_1 = 11.1$ A as well as $I_2 = 12.6$ A. We think that this type of scenario is realistic because different applications showed different power profiles. When we want to start a job D now, we have the problem of no nodes being available because M_1 and M_2 are both too close to the power limit to start the job. Although on the combined system there are both free nodes and a margin of power available, we can not start a new job because the power balance between both partitions is not equalized for this particular job configuration.

Solution Our solution is based on the assumption that we are supplied with two wall plugs with in-phase AC currents. We are then able to deploy a self-made cable that enables us to use the circuit shown in Figure 2 to power our cluster. We checked the phases using an ordinary two channel oscilloscope before the challenge.

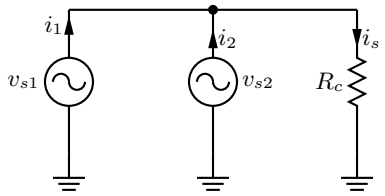

Fig. 2. Power Equalization

It is obvious that—provided the two voltages are in-phase—the configuration of M_1 and M_2 is not important, so the whole cluster and components like the Myrinet switch can be represented by R_c and it thus follows from these conditions that $I_1 = I_2 = I_s/2$, so both ammeters measure the same current.

During the challenge, we experienced a slight difference between both currents. Lacking another ammeter we could not verify the reason for this, but we strongly believe that measurement inaccuracies in one (or both) units have been causing that effect since we observed a nearly perfect correlation between the two currents.

2.2 CPU Frequency Scaling

Introduction Since energy efficiency is becoming an increasingly important feature for mobile devices and server systems [4–6], most of the currently available

processors include features for power management, such as frequency scaling [7] or dynamic frequency and voltage scaling [8]. All these features provide different modes of operating the processor. With these different modes of operation it is possible to trade performance for energy efficiency, by employing one of the techniques mentioned above.

Although one has to choose the performance mode carefully [9], previous research has shown that power dissipation is proportional to clock speed on most modern general purpose CPUs and mostly determined by the underlying chip technology rather than the microarchitecture [10].

However, it is clear that not all applications are affected by CPU frequency scaling in the same way; for example, memory bandwidth limited applications can not benefit from a clock frequency much higher than the memory bus bandwidth. A performance study carried out by IBM showed that STREAM Triad [11] benchmark results do not vary significantly above 1.8 GHz on an Opteron equipped IBM eServer 325 while LINPACK [12] performance scales almost linearly with the CPU frequency on the same machine [13]. We can confirm this behavior with our own measurements shown in Table 2. Those measurements had been performed on a IBM System x iDataPlex dx340 system which is the predecessor of the dx360 system using the same CPU and energy efficient design. For the measurements in this table we used the following HPL parameters: $N = 40000$, $NB = 200$, $P = 4$, $Q = 8$ and used all 32 cores of 4 nodes.

Frequency scaling in a power-limited environment All applications used in this years cluster challenge (see Section 2.3 for power dissipation profiles for RAxML and WPP) show different behavior in terms of power consumption. It was clear that, given the 26 A current limit imposed by the competition rules, we could use more nodes for the WPP, OpenFOAM, GAMESS and POY runs. Nodes running HPCC and RAxML used more power, so we had to run these applications on a smaller number of nodes. Since the competition rules also stated that every piece of equipment that is used for calculations at one point in time at the competition (except spare parts) had to be powered all the time, the number of nodes used for example for HPCC can be modelled by the equation

$$(n - x)p_i + xp_l \leq 26 \text{ A} \tag{1}$$

where n is the total number of nodes used for the competition, x is the number of nodes running HPCC, p_i specifies the current drawn by a single node when idle, and p_l when running HPCC (both measured in amperes).

Since we measured that we could leverage $n = 14$ nodes with some of the applications, we would end up with $x = 6$ nodes for HPCC—but that would imply the waste of approximately 9.5 A of current during the HPCC run because of 8 nodes being idle during that time, if we ran at the default CPU frequency of 2.5 GHz.

The Linux kernel used already issues `HLT` instructions [14] when the system load is low, thus, the idle power is hard to reduce further without hardware modifications (i.e., reducing the amount of memory), which in turn might deteriorate

the performance of other applications, or more invasive kernel modifications such as decreasing the timer interrupt frequency. However, based on the linear scaling of LINPACK performance with the CPU clock frequency, it seemed promising to reduce the clock frequency of the nodes running HPCC, so that we could run this particular benchmark on more nodes with less power dissipation per node. Applying this technique reduces the amount of power wasted by idle nodes and yields a slightly higher performance as shown in Figure 3. The graph in this figure is based on Equation 1 and the measurement results from Table 2.

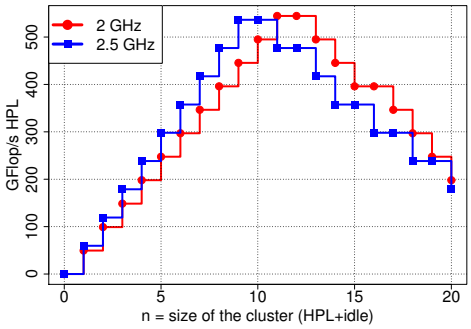

Fig. 3. Modelled influence of frequency scaling on LINPACK performance for different cluster sizes n with a fixed current limit of 26 A

Table 2. Effects of frequency scaling on HPL on a single dx340 node

CPU Frequency	2.0 GHz	80%	2.5 GHz	100%
Current Idle p_i	1.0 A	100%	1.0 A	100%
Current Load p_l	2.25 A	82%	2.75 A	100%
Gflop/s	49.5	83%	59.67	100%

Results While this approach worked very well on our IBM System x iDataPlex dx340 based test systems, it turned out that, because of problems in the ACPI subsystem with the default BIOS, CPU frequency scaling was not supported on the system used at the challenge, which consisted of IBM System x iDataPlex dx360 nodes. After reporting this problem to our vendor partner IBM we were issued a pre-release BIOS which supported ACPI well enough so that we could query and "adjust" the clock frequency with the acpi-cpufreq driver and the cpufrequtils software package. Unfortunately, the adjustments had no measurable effect on the power consumption of our system. A micro-benchmark—based on continuous sampling of the processors internal time stamp counter—revealed

that the CPU frequency did not change at all, even though the appropriate entry in `/proc/cpuinfo` showed different values than before. Even with on-site support from an IBM technician, this problem could not be solved until the start of the challenge. However, this issue only exists on our particular platform and is clearly firmware related, and so the analysis described above is still useful in similar scenarios.

2.3 Node Interconnection Network

Our vendor partner IBM provided us with 14 Mellanox ConnectX InfiniBand cards (MT26418) and a Cisco TopSpin SFS7000D switch. Myricom kindly offered to also sponsor us with their own latest high performance interconnect technology, fiber based 10G-PCIE-8B-QP Myricom cards and a switch from Myricom with a single line card (10G-SW32LC-16QP). Therefore, we could decide which interconnect we wanted to use. It was quickly discovered that Myrinet was much easier to setup on the cluster. All it took after plugging in the cards was a simple `./configure && make && make install` to build the driver and `mx_local_install` on all nodes to get a usable system. The driver installation was more complicated with InfiniBand, although the OFED software stack was officially supported by the SuSE Linux SLES 10.2 distribution we used.

Furthermore, an evaluation of the performance and power consumption of both networks should be the basis for the final decision on the interconnect. The results of this analysis are described in the following paragraphs. We compare the two most widely used high performance interconnection networks for cluster computing, Myrinet and InfiniBand, and analyze them with regards to their micro-benchmark performance, real application performance and power consumption.

InfiniBand [15] is the most-used commodity high performance network in cluster computing. It offers bandwidths of up to 32 Gb/s. The latency can be lower than 1 μs if it is measured in a tight communication loop. The InfiniBand Architecture (IBA) has been analyzed in many research works [16, 17] and is well understood. There also exist two major open source Message Passing Interface (MPI) [18] implementations for InfiniBand, MVAPICH and Open MPI.

Myrinet [19] is the 4th generation Myricom hardware. It offers a bandwidth of 10 Gb/s for either Myrinet Express (MX) or Ethernet protocols. Latencies down to 2.3 μs are possible and its physical layer is 10 Gigabit Ethernet. The MX communication layer is highly optimized for MPI point-to-point messaging and offers dynamic routing [20]. Myrinet is able to perform tag matching in the NIC firmware which further offloads communication functionality from the main CPU.

Different research groups compared the performance of communication networks. Liu et al. [21] compares the characteristics of several high performance interconnection networks with micro-benchmarks. Another study by the same author [22] also presents some application comparisons. Other studies, like [23–25] limit themselves to micro-benchmarks and the NAS parallel benchmarks. However, for the decision which network technology to use for Cluster Challenge

it was crucial to not only compare micro-benchmark results but also real-world application performance and power consumption.

Micro-benchmark Results The well-known benchmark NetPIPE [26] was used to measure basic parameters such as latency and throughput. Figure 4 shows the latency for small messages. The minimum (zero-byte) latency for InfiniBand with MPI was 1.38 μs and 2.53 μs for Myrinet. Figure 5 shows the throughput that NetPIPE reported for the different networks. The highest throughput with 8 MiB messages was achieved with InfiniBand at about 13.9 Gb/s which is 86.9% of the peak performance. Myrinet reached up to 9.1 Gb/s which is 91% of the peak bandwidth.

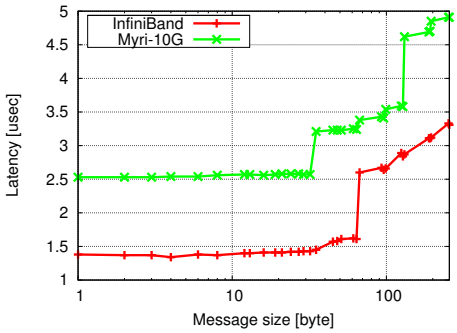

Fig. 4. Latency for small messages

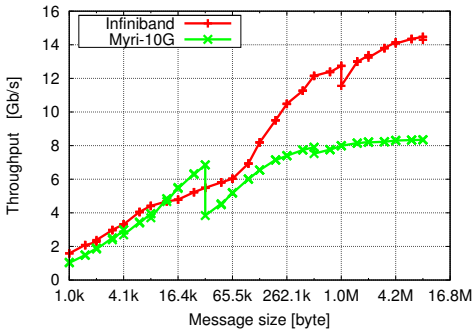

Fig. 5. Throughput for large messages

Application communication In this section, the different networks will be compare with respect to application performance, using the same input problem and an identical system configuration (except the communication network) for each application run. The total run time and the different MPI overheads are recorded. We ran each application three times and report minimal values in order to eliminate operating system noise effects. The PMPI [18] interface was leveraged to intercept and profile all MPI calls and report the average overheads over all processes.

The MPI parallelization of RAxML (see Section 3.5) is coarse grained: each rank computes different trees and sends the results to rank 0. We calculated 112 phylogenetic trees on all 112 cores of our cluster with both networks. We used the same random seed for all runs and a database with 50 pre-aligned genome sequences consisting of 5000 base pairs: this database is shipped with the RAxML source. This computation took 746.97 s with InfiniBand and had an MPI overhead of 34.90 s (4.67%). On Fiber Myrinet 738.35 s were needed with an MPI overhead of 31.60 s (4.28%). Most of the MPI overhead is contributed by `MPI_Probe`. The MPI overhead for both networks is shown in Figure 6.

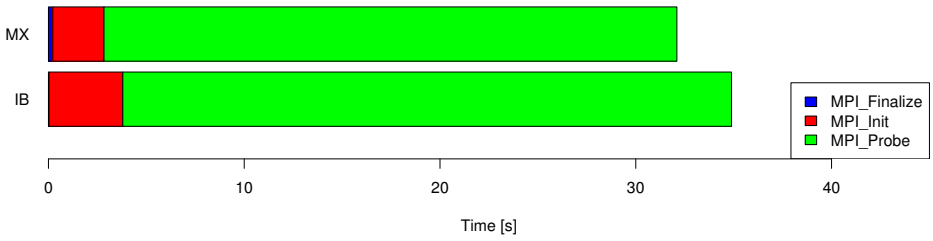

Fig. 6. MPI communication overhead of RAxML

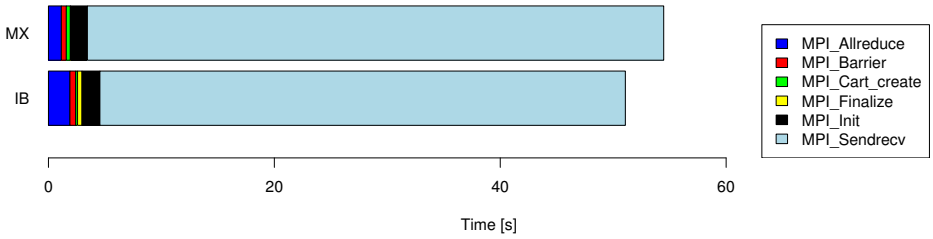

Fig. 7. MPI communication overhead of WPP

Our WPP (see Section 3.6) benchmark simulated a grid of size $30000 \times 30000 \times 17000$ with a spacing of 20 and a single wave source in it on all 112 cores. The layout of the grid was the same as in the LOH1 example distributed with WPP. All output was written after the last time step was computed. The MPI overhead of WPP is dominated by `MPI_Sendrecv`. On InfiniBand, the calculation took 701.60 s and showed an MPI overhead of 51.08 s (7.28%). On Myrinet, the timing was 700.95 s of which 53.37 s (7.61%) had been recorded as MPI overhead. An overview of the MPI calls that contribute to WPP's MPI overhead can be found in Figure 7.

Power Measurements In this section, we analyze the power consumption of the different applications and cluster configurations. This was done by sampling the root mean square current through the whole cluster (described in Section 2) every second. The power consumption can easily be computed by multiplying this value with the root mean square voltage (120 V in our case). However, we will give our direct measurement results (in amperes) throughout this section. With this method, we were able to compute the total power consumption for the solution of a particular problem for each application. In this case, the power consumption is the discrete integral (sum) over all measurement points multiplied by 120 V. The current graphs over time and the total energy needed to compute a particular input are reported.

In a first experiment, we compare the current drawn by our idle system (without the switch) with the three different network configurations. The system equipped with InfiniBand uses 17.7 A when idle. Myrinet lowers the current

consumption of the idle cluster to 16.9 A. Disconnecting all Ethernet cables from the on-board Gigabit Ethernet (GigE) cards lowers the current by 0.35 A. Since Myrinet as well as InfiniBand also offer IP connectivity it was clear that (additionally) using Gigabit Ethernet for the challenge would be a waste. By deciding to not use GigE at all we also saved the power that would have been consumed by the GigE switch. Furthermore, the current consumption of the switches was analyzed. The Cisco InfiniBand switch uses 0.48 A. The 7U Myrinet switch uses 0.75 A with the fan unit. However, removing the fan decreased the drawn current to 0.48 A. The Cisco Catalyst Gigabit Ethernet switch drew 0.6 A of current.

The current consumption of all switches was identical when idle and under full load. We also investigated the current consumption of four nodes under full communication load using a bidirectional stream of 8 MiB messages. Four InfiniBand nodes used 3.9 A when idle which increased to 5.0 A under full message load. Four Myrinet (copper) nodes used 3.77 A when idle and 4.95 A under full load with Open MPI using the OB1 PML. However, switching to the PML CM, which enabled tag matching in hardware, reduced the current by 4% to 4.75 A. This seems mostly due to the packet matching on the specialized NIC processor.

In the following, the power consumption of each application and input problem for the different networking technologies will be compared. Furthermore, we compute and compare the total energy consumption that is required to solve the particular real-world problem. For this we only compare two applications, WPP and RAxML. This is due to the fact that we were also interested in profiling the MPI overhead of the applications. Since GAMESS is faster when sockets are leveraged for communication and the POY runtimes varied too much even for identical input problems we omit results for those applications. Unfortunately, at the time we conducted our measurements we did not have enough experience with the parallel solvers of OpenFOAM to produce useful results.

The RAxML computation has different phases with different power consumption. This demonstrates that the CPU is used differently in those phases. One would assume that a higher power consumption means more efficient CPU usage. RAxML has a peak with more than 35 A in our measurement. The generation of the “tree of life” for the 50 species in our input file used 8.315 kWh in InfiniBand. Myrinet uses 8.015 kWh which is around 3.7% less than InfiniBand. RAxML only exhibits a small communication overhead, thus the energy consumption is only marginally influenced by the network.

The Wave Propagation Program uses between 29 A and 31 A and the power consumption varies highly during the application run. The computation of the seismic properties modelled in the (modified) LOH1 example shipped with WPP uses 6.807 kWh on InfiniBand. Myrinet lowers the energy consumption by 1.4% to 6.713 kWh.

Conclusions The first and most important conclusion is that networking micro-benchmarks and simple metrics such as latency and bandwidth do not necessar-

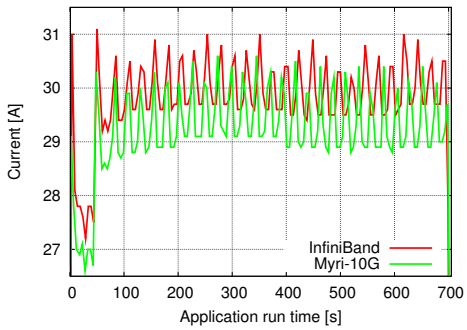

Fig. 8. Current consumption for WPP

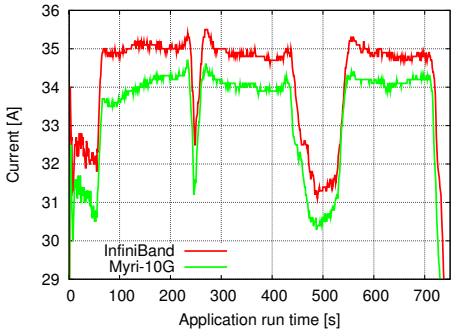

Fig. 9. Current consumption for RAxML

ily reflect the performance of real-world applications—if we had decided which interconnection network to use for the challenge by solely relying on micro-benchmarks, we clearly would have ended up with InfiniBand. Many other effects such as support for tag matching in hardware, memory registration or remote direct memory access influence the performance of real applications significantly. We demonstrated that even though micro-benchmarks predict that Myrinet should be slower than InfiniBand, Myrinet performs significantly better than InfiniBand for some examined applications. Power consumption is also an important parameter for high performance networks, especially for the Cluster Challenge where we were faced with a hard power limit. We, furthermore, demonstrated that the energy needed to compute a certain result can be decreased with a power-efficient interconnection network. We also show that the energy consumption of an idle system significantly depends on the networking equipment. Based on our findings we decided to work with Myrinet cards during the challenge, which proved to be a good decision in terms of performance as well as stability and ease of use.

2.4 Hard Drive Suspends

In order to save additional energy, we decided to boot cluster nodes diskless and enable the local hard disks only when needed. Most hard drives feature a suspend mode where the disk stops spinning and the head is in a waiting position. That behavior can be triggered using the hdparm utility supplied with most Linux distributions.

We created a custom initrd boot image which contained the Myrinet kernel drivers and userspace stack. The initrd images were deployed onto USB flash drives. During the boot process, the head node's root file system is mounted via NFS. This procedure enabled us to completely discard the Ethernet while still being able to benefit from diskless booting. We used 13 USB flash drives of 2 GB size, one for each compute node. Measurements showed that the flash drives consume less than 0.2 A for the whole cluster when not used after the boot procedure.

This setup allowed us to choose between a completely diskless mode of operation and a mode with plenty of scratch disk space on a per-job basis which saved 1 A when all hard disks are suspended. Further research could determine if there are performance issues in communication intensive applications related to Open MPI writing files to the `/tmp` directory. We did not encounter any significant decrease of communication performance with the applications used in the competition, though.

2.5 Job Scheduling

The main strategy for the challenge was to maintain a constantly high level of workload to make sure there is no unused CPU time. We also tried to reduce communication overhead. This involves a couple of strategies such as avoiding job oversubscription, i.e. allocating a core for multiple jobs, and reducing the ratio of nodes to threads, i.e. bundle threads of one job on the fewest nodes possible. Another point taken into consideration was the scalability of the applications; in spite of low speedup it was necessary to increase the number of cores allocated to a particular job to ensure that the job terminates within the time limit. Based on the scheduling strategy of the previous years winning team [27], we developed a new strategy that takes into consideration constraints like time limit, points per job, power consumption, etc. We created an abstract mathematical model based on a 0-1 knapsack problem to solve the problem of getting as many points as possible in the given time. Our MATLAB script uses the solver for the constrained binary integer programming problem

$$\max_{x_i \in \{0,1\}} \sum_{i=1}^{n} p_i x_i \,, \qquad A \cdot x \leq b \,.$$

Each given job x_i is associated with its score points p_i. The constraints A consist of CPU time and energy for each job and b holds the challenge constraints, the first component being the overall time for the challenge, the second being the maximum energy $E_{max} = U \int_{t_s}^{t_e} I(t)\mathrm{d}t$ being provided to the participating teams for the duration of the challenge (t_s being the start time of the challenge, t_e the end time, I being $I_1 + I_2$ according to section 2.1) and U representing the nominal system voltage e.g., 120 V for the US. For the example

$$p = \begin{bmatrix} 2 \\ 6 \\ 3 \end{bmatrix}, \quad A = \begin{bmatrix} 5\text{ h} & 10\text{ h} & 5\text{ h} \\ 6\text{ kWh} & 18\text{ kWh} & 7\text{ kWh} \end{bmatrix}, \quad b = \begin{bmatrix} 12\text{ h} \\ 15\text{ kWh} \end{bmatrix}$$

the solution computes to $x = [1, 0, 1]^T$ because the 18 kWh of the second job exceeds the limit of 15 kWh and only the first and third job would be started. By calculating the estimated CPU-time-per-point rate for each job, the result can be converted into two different priority lists: one list for runnable jobs and one for not-runnable, reserve jobs. Those lists are the initial solution for a modified

version of the hill-climbing algorithm for oversubscribed scheduling [28]. The algorithm uses the current state and configuration of the cluster and the challenge limitations to find the next job.

Although suitable for the work loads encountered in the challenge, the model has some limitations. Each job's power consumption is represented by a constant, taking into account the average power consumption during its run time. As mentioned in Section 2.3, applications' power consumption changes with time. Further, single jobs are—under certain circumstances—allowed to exceed power limitations, especially when occupying the whole cluster. Although this could be filtered by the hill climbing algorithm, we did not implement a check, as those conditions are rare and trivial to recognize.

Unfortunately—as opposed to the teams in the 2007 challenge—we did not receive a list of score point values for the jobs at the SC08 Cluster Challenge, so this strategy was not viable. The new strategy for the beginning, after an analysis of the given jobs, was to split the cluster into one bundle of 8 nodes for larger jobs that can't run on one node, and another bundle for single node jobs. Once we reached a point in the competition where no more jobs would terminate before the deadline, we scheduled jobs that write intermediate data so that we could get partial points.

3 Application Optimization

The SC Cluster Challenge Committee chose HPC applications from different fields of scientific computing. GAMESS is used to simulate processes in molecular chemistry, OpenFOAM simulates different physical phenomena including fluid dynamics, combustion and electrostatics, but is also able to simulate financial flows, and WPP implements algorithms to predict wave propagation in earth quakes. RAxML and POY both originate in the field of biogenetic simulation, utilizing phylogenetic trees to provide insight into evolutionary processes.

Since all applications have been published under open source licenses, we were able to begin deployment and source code analysis on the clusters at ZIH as soon as the names of the applications were announced. We started with simple profiling utilizing gprof in order to obtain a basic overview about the time spent in different functions. VampirTrace and Vampir [29] allowed a better understanding of communication patterns and were utilized together with PAPI counters to find regions of code with potential for architecture specific code optimization.

3.1 HPCC

The High Performance Computing Challenge benchmark [3] is one of the worldwide accepted benchmarks for comparison of HPC systems. Seven micro-benchmarks are included in the test suite, which partially split up into parallel, embarrassingly parallel and single measurement kernels. The micro-benchmarks are HPL [12], DGEMM, STREAM [11], PTRANS, RandomAccess, FFT and bandwidth/latency for the network.

Analysis The rules for HPCC runs are strict: modification of source code is not allowed in the base run and only a few changes in some files are permitted for the optimized run. With respect to this, VampirTrace was only used to get an overview of the fraction of communication in the different kernels.

Optimization The task of optimization consists of three distinct parts: Finding the best compiler and compiler flags, using the fastest libraries on the specific machine and building an input file to maximize performance based on machine specifications and time limit. We compared different compiler suites, namely the Intel Compiler Suite Professional Edition for Linux (version 10.1), the GNU Compiler Collection (version 4.3) and Portland Group's PGI Compiler Suite (version 7.2-5). The Intel compilers showed the best performance in our tests. In the field of linear algebra the Intel Math Kernel Library (version 10.1) was a few percent slower than GotoBLAS (version 1.26).

Listing 1.1 shows the compiler flags we used to compile HPCC. To maximize memory bandwidth for the STREAM benchmark, the flags shown in Listing 1.2 were added.

Our efforts to optimize HPCC's FFT micro-benchmark showed that it is possible to achieve a 60 % runtime improvement for the FFT kernel by linking against a third party FFT library. This kernel calculates a double-precision complex one-dimensional DFT. There are extensive performance comparisons based on similar hardware [30] for different FFT libraries available. Unfortunately, the interface of the FFTW 3.2 library is incompatible to the HPCC benchmark, so we used the FFT routines of the Intel MKL. We had to apply minor changes to the MKL interface to link the benchmark against FFTW.

Work on finding a good combination of input parameters was done by Stewart et al. [31], so we had a starting point for individual tuning. We used the parameters from Listing 1.3 for the run that was scored at the start of the challenge.

Listing 1.1. HPCC compiler flags

```
-DUSING_FFTW -DRA_SANDIA_OPT2 -DHPCC_MEMALLCTR -xS -axS -O3 \
-fno-alias -ansi-alias -no-prec-div -no-prec-sqrt -restrict \
-fomit-frame-pointer -funroll-loops -ip -Zp16 -malign-double
```

Listing 1.2. Additional STREAM compiler flags

```
STREAMCCFLAGS = -fno-alias -opt-streaming-stores always
```

Listing 1.3. Relevant lines from the `hpccinf.txt` we used

```
112640   Ns
220      NBs
8        Ps
8        Qs
0        RFACTs (0=left, 1=Crout, 2=Right)
0        BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
```

```
0        DEPTHs (>=0)
96       swapping threshold
16       memory alignment in double (> 0)
```

For the HPCC run completed at the challenge, eight nodes (64 cores) were used. Our optimization, i.e. compiler flags, libraries and input parameters, yielded significant performance gain in comparison to the first test runs as shown in Table 3.

Table 3. HPCC micro-benchmark results in comparison to initial results

Benchmark	Value (challenge run)	Performance gain
HPL	526.3 Gflop/s	$\approx 6\%$
MPIFFT	15.581 Gflop/s	$\approx 60\%$
StarSTREAM Triad	1.030176 GiB/s	$\approx 45\%$
MPIRandomAccess	0.129382439 GUP/s	$\approx 20\%$

3.2 General Atomic and Molecular Electronic Structure System (GAMESS)

The application is a general ab initio quantum chemistry package. GAMESS-US [32] (hereafter referred to as GAMESS) is able to compute a wide range of self-consistent field molecular wave functions. Nuclear gradients can be used for automatic geometry optimization, transition state searches, or reaction path following. Many other computations are available, including calculation of analytic energy gradients, energy hessians, third order Douglas-Kroll scalar corrections, and various spin-orbit coupling options. The main parts of GAMESS – I/O management and calculation – are written in FORTRAN77. The parallelization of GAMESS started in 1991 with the use of the TCGMSG [33] library. The TCGMSG library was replaced by the Distributed Data Interface (DDI) [34], which sets a layer between the computation and communication to have a fixed interface for the chemistry algorithms and an exchangeable communication library. Supported communication libraries are sockets, MPI, SHMEM, LAPI and ARMCI.

Analysis Profiling and tracing with gprof and VampirTrace showed GAMESS having no real hot spots in code. We compared four different types of communication, in detail sockets, mixed mode (sockets for intranode, MPI for internode communication), MPI and ARMCI, which were usable for our system configuration. For shared memory systems GAMESS forks the number of compute processes specified in the run command. When using distributed memory systems like HPC clusters, GAMESS creates one additional data server for each compute process. When executing internode test runs, the MPI or mixed mode

version was twice as slow as the sockets version of GAMESS, because the data servers are in a sleep mode for the sockets version and only work when needed while the data servers for MPI are constantly spinning for requests. If there is one compute process on each core, in MPI mode, the additional data server oversubscribes the node and decreases performance. One solution for this problem is to use only half the cores for computation, but that would waste CPU time. ARMCI [35], which simulates a shared memory system, was as slow as the MPI version.

Another difficult aspect of running GAMESS are temporary files, which are written by each compute process and can grow up to several GiB in size. The presentation by B.J. Lynch [36] includes an example GAMESS run showing the utilization of a local hard drive being 40% faster than an NFS mounted directory.

Optimization Most modifications of source code to gain performance result in erroneous runs with incorrect output. The check routine reports variations in the results for the test cases delivered with GAMESS. Therefore source modifications were discarded. With regard to temporary files, we could not use RAM disks on the nodes since the cluster had not been equipped with sufficient memory to both allocate memory for GAMESS and create scratch filesystem on a RAM disk. The workaround was to use suspended local disks as described in Section 2.4.

Application tuning with compiler flags was done under the premise of all examples provided by GAMESS passing the built-in correctness check. As a first attempt the source code was compiled with aggressive compiler flags for the Intel Fortran Compiler

```
-xS -axS -O3 -fno-exceptions -fno-instrument-functions
-funroll-loops -ip
```

producing incorrect computation results. To eliminate numerical inaccuracies, several objects must be generated applying `-O2` instead of `-O3`. Additionally, the use of `-fp-model precise` was necessary for a limited set of source files, e.g. `eigen.src`. This optimization brought us a performance gain of approximately 5%–15%. This depends on the input data and thereby on the type of calculation, with its specific functions. For example the `zeolite` dataset, one of the last years input data, was 9% faster with optimization.

3.3 Open Field Operation and Manipulation (OpenFOAM)

OpenFOAM [37] takes a very general approach to solving various problems that can be described using partial differential equations. OpenFOAM itself defines a framework that helps scientists to implement problem specific solvers. Developers made extensive usage of object-oriented features of the C++ language, e.g., inheritance, polymorphism, and class templates.

Analysis Since automatic instrumentation done by the compiler cannot be used to trace code inside shared libraries with VampirTrace [38], we had to write

extensive additions to the supplied build system. Trace size proved to be another problem caused by millions of function calls, filling the memory segments and ruining the trace with interrupts caused by frequent flushes. A combination of filters applied at runtime and short trace runs enabled us to make significant statements on communication patterns and hot spots, but it also degraded the quality of those statements.

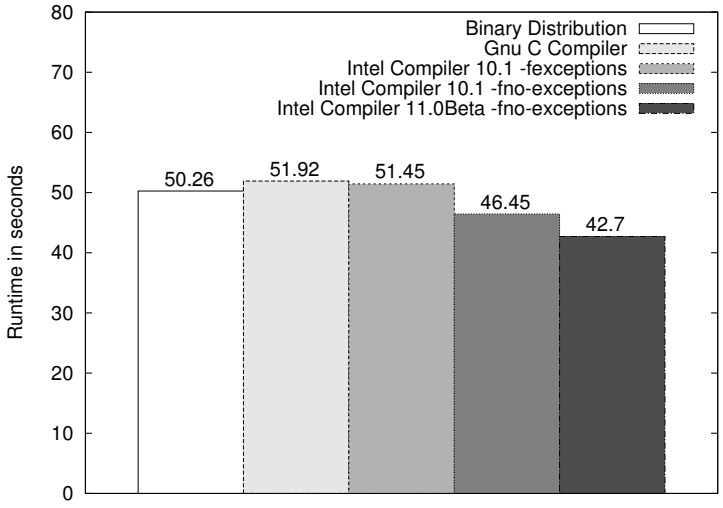

Fig. 10. OpenFOAM performance results, damBreakFine case, 1 core

Optimization We had to modify[3] the source code to compile with Intel's C++ Compiler which is known to optimize numerical code very well [39] on x86 systems. Although exceptions can rarely be found in the OpenFOAM source code, we suspected some potential for optimizations in eliminating them. The resulting code[4] compiles with icpc's `-fno-exceptions` flag and crashes in case of an internal error that would otherwise have been caught and processed. We encountered this neither in our preparation tests nor during the challenge. Figure 10 shows that these modifications caused a significant decrease in run time, providing us with a significant advantage over the other teams, while automatic compiler optimizations like Interprocedural Optimization which are designed to optimize this sort of code [40] had no notable effect on OpenFOAM's runtime performance.

[3] http://wwwpub.tu-dresden.de/~jmuelle/of_icc.patch.bz2
[4] http://wwwpub.tu-dresden.de/~jmuelle/of_no-exceptions.patch.bz2

3.4 Phylogenetic Analysis of DNA and other Data using Dynamic Homology (POY)

POY [41] is a phylogenetic analysis program for molecular and morphological sequences used to study evolutionary relations among species or populations of organisms by building a "relationship tree" that shows their probable evolution. POY implements the concept of dynamic homology allowing optimization of unaligned sequences and supports algorithms like multiple random addition sequence, swapping, tree fusing, tree drifting and ratcheting. It also allows the analysis of entire chromosomes and genomes, taking into account large-scale genomic events (translocations, inversions and duplications).

Analysis POY is written in Objective Caml (OCaml) and C. OCaml is a multi-paradigm programming language supporting functional, imperative and object-oriented programming. Profiling the execution of a non interactive analysis with gprof provided not much insight for eventual optimization approaches. Aside from two functions belonging to garbage collection no particular function accumulated more than 5% of the run time.

Fully analyzing our test runs with VampirTrace and Vampir would have required instrumenting not only POY itself but also OCaml. On the other hand, for evaluating just POY's communication structure it was sufficient to use VampirTrace's automatic instrumentation capabilities only on POY. Admittedly the structure of POY's MPI communication appeared too complex to optimize it with the time and effort we could devote to it.

Using a sophisticated programming language like OCaml may facilitate development of scientific applications. Then again, it makes performance analysis and optimization very aggravating for people not versed in its peculiarities. Regarding POY we found it difficult to evaluate its overall performance, so we decided not to spend additional time on optimization attempts.

3.5 Randomized Accelerated Maximum Likelihood (RAxML)

RAxML [42] is a phylogenetic analysis software which calculates a tree of life of some organisms using their gene sequences. As its name implies, RAxML uses randomized search heuristics. There are three different versions of RAxML:

- Single-threaded without parallelization
- Parallelized using POSIX Threads for shared memory systems
- MPI parallelized for distributed memory systems

Since the MPI version is parallelized in such a way that each rank computes a different phylogenetic tree we could not use this variant for Cluster Challenge, where we had to compute only one tree for each dataset. Therefore we used the RAxML version which leverages POSIX Threads to do a fine grained parallel tree inference.

Analysis Profiling RAxML using gprof instantly revealed that a special set of functions whose names begin with `newview`, e.g. `newviewGTRGAMMA` and `newviewGTRCAT` consume about 70% of a typical job's runtime. Further analysis with Vampir showed that most of the algorithms distribute their workload over several threads efficiently. It is worth noting that the MPI master thread is idle most of the time, its main purpose being distributing work and writing results. Thus, there is a note in RAxML's manual that the program should be executed with one more MPI process than physical CPUs (cores) available. This communication structure may cause delays for worker threads, at the beginning of a program run.

Altough several `MPI_Send` calls could have been replaced by `MPI_Broadcast` we decided not to change the communication structure because, with growing input data size (sequence length), the computing time increases rapidly while communication overhead does not. When considering the entire run time the impact of MPI communication becomes marginal.

Optimization Our attempts to tune the `newview` functions described above have not been successful. Reading PAPI hardware counters showed that the Intel Compiler was already making intense and efficient use of SSE instructions within these functions. For RAxML, customizing compiler flags to fit our architecture was the only optimization we applied for the binaries used during the challenge.

3.6 Wave Propagation Program (WPP)

WPP is a comprehensive wave propagation simulation program. Simulating seismic waves caused by earthquakes or explosions (the main purpose of the software) is important for many scientific fields; for instance the search for oil, the construction of underground facilities, or the calculation of possible earthquake effects on a big city. One of the advantages of WPP compared to most other seismic simulation tools is the special treatment of complex geometries. Traditional finite element simulations in this field tend to be less accurate in case of a very complex simulation (not box shaped, not in the direction of the grid shape). WPP is able to utilize local mesh refinement to simulate local, high frequency waves in higher detail.

As one result of our work we can state that WPP is a highly-optimized tool which yields most of the computational power an x86 CPU can deliver. Nevertheless, it was possible to apply optimizations which significantly reduce run time.

Analysis A first analysis has shown that there are two major runtime consumers. First, functions from the external Blitz++ library with a high call count and, second, one of the core functions of WPP, `inner_loop_5()` with a low call count but long computational runtime. In a first Vampir analysis we learned that there is a initialization phase of basically the same runtime for each input file. A second phase is the actual calculation which consists of alternating blocks of

Blitz++ functions and `inner_loop_5()`. Finally there is a short end phase with constant runtime. Hence, the two hotspots which could significantly influence runtime were analyzed more deeply. With use of PAPI counters we determined that `inner_loop_5()` is already well-optimized with a low cache miss rate and a high operation per second count. By looking at the call graphs of the Blitz++ blocks we saw a relatively high number of thrown exceptions and a very high call count for some simple functions.

Optimization Because of the knowledge we gained from the analysis we decided to first optimize WPP via Intel's C/C++ Compiler and specially selected compiler flags. For compiling with icc it was necessary to fix some non-standard source code in sections of third party libraries. For best performance, Blitz++ needed to be compiled without exception handling and with aggressive inlining options. This decreases the runtime of the WPP example case `Layer.in` by about 20%. The `inner_loop_5()` function compiled with icc and aggressive loop unrolling decreased its runtime by about 20%.

4 Conclusions

In this paper we described our approach to tune a commercially available cluster system in order to minimize power usage and maximize time efficiency when running workloads from real world scientific applications. We modified both software and hardware to create a system that performed better than the competition.

The usage of state-of-the-art analysis tools like Vampir allowed us to visualize application behavior and to delve into the code where we saw potential performance gain. Although software supported analysis made it very easy to achieve results, the optimization itself still required hard work and creative ideas.

By being able to do all of this in advance of having access to the hardware we used in the challenge, we could concentrate on hardware optimization as soon as it was possible. As described in Sections 2.2, 2.3 and 2.4, we succeeded in stripping the system of anything we did not essentially need, for example the complete Gigabit Ethernet network, the Myrinet switch's fan and half of the system's RAM. Not only did we save power, in real world scenarios we also would have saved money we otherwise would have spent on hardware which we did not need and which also would have increased the system's total cost of ownership by consuming power to no avail. This shows that purchasers of a cluster system can benefit from carefully considering the requirements of their applications and order just what they need.

Acknowledgments

First of all we want to thank our sponsors Myricom and IBM. Without their generosity an event like Cluster Challenge could not be successful. We also want to thank ZIH (TU Dresden) and UITS (Indiana University) which offered some

of their own hardware so that we could work on clusters similar to our challenge system before the actual challenge. Thanks to Matt Link, Robert Henschel, Jeff Gronek, Richard Knepper, and Ray Sheppard for all their support during the challenge preparations. We also want to thank Jeremiah Willcock for helpful comments on the paper.

References

1. Wu-chun Feng and Kirk Cameron, "The Green500 List: Encouraging Sustainable Supercomputing," *Computer*, vol. 40, no. 12, pp. 50–55, 2007.
2. Energy Information Administration, "October 2008 Mothly Energy Review (MER)," October 2008. `http://www.eia.doe.gov/emeu/mer/pdf/mer.pdf`.
3. P. R. Luszczek, D. H. Bailey, J. J. Dongarra, J. Kepner, R. F. Lucas, R. Rabenseifner, and D. Takahashi, "The HPC Challenge (HPCC) benchmark suite," in *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, (New York, NY, USA), p. 213, ACM, 2006.
4. J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 103–116, ACM New York, NY, USA, 2001.
5. P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in Web servers," *Power Aware Computing*, 2002.
6. Y. Shin, K. Choi, and T. Sakurai, "Power optimization of real-time embedded systems on variable speed processors," in *Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design*, pp. 365–368, IEEE Press Piscataway, NJ, USA, 2000.
7. K. Farkas, J. Flinn, G. Back, D. Grunwald, and J. Anderson, "Quantifying the energy consumption of a pocket computer and a Java virtual machine," in *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 252–263, ACM New York, NY, USA, 2000.
8. M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Kluwer International Series in Engineering and Computer Science*, pp. 449–472, 1996.
9. A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar, "Critical power slope: understanding the runtime effects of frequency scaling," in *Proceedings of the 16th international conference on Supercomputing*, pp. 35–44, ACM New York, NY, USA, 2002.
10. R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277–1284, 1996.
11. J. McCalpin, "Memory bandwidth and machine balance in current high performance computers," *IEEE Technical Committee on Computer Architecture Newsletter*, pp. 19–25, 1995.
12. J. Dongarra, "The LINPACK benchmark: An explanation," *Evaluating Supercomputers*, pp. 1–21.
13. D. Pase, "Performance of the IBM® 325 for scientific and technical applications,"
14. E. Rohou and M. Smith, "Dynamically managing processor temperature and power," in *2nd Workshop on Feedback Directed Optimization*, 1999.

15. The InfiniBand Trade Association, *Infiniband Architecture Specification Volume 1, Release 1.2*, 2004.
16. G. M. Shipman, T. S. Woodall, R. L. Graham, A. B. Maccabe, and P. G. Bridges, "InfiniBand Scalability in Open MPI," in *Proceedings of IEEE Parallel and Distributed Processing Symposium*, April 2006.
17. J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High performance RDMA-based MPI implementation over InfiniBand," in *ICS '03: Proceedings of the 17th annual international conference on Supercomputing*, (New York, NY, USA), pp. 295–304, ACM, 2003.
18. MPI Forum, "MPI: A message-passing interface standard, version 2.1," September 4th 2008. `www.mpi-forum.org`.
19. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, "Myrinet: A gigabit-per-second Local Area Network," *IEEE Micro*, vol. 15, no. 1, pp. 29–36, 1995.
20. P. Geoffray and T. Hoefler, "Adaptive Routing Strategies for Modern High Performance Networks," in *16th Annual IEEE Symposium on High Performance Interconnects (HOTI 2008)*, pp. 165–172, IEEE Computer Society, Aug. 2008.
21. J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. Kini, D. K. Panda, and P. Wyckoff, "Microbenchmark performance comparison of high-speed cluster interconnects," *IEEE Micro*, vol. 24, pp. 42–51, January 2004.
22. J. Liu, B. Chandrasekaran, J. Wu, and W. Jiang, "Performance Comparison of MPI implementations over Infiniband, Myrinet and Quadrics," in *Proceedings of International Conference on Supercomputing*, 2003.
23. J. Hsieh, T. Leng, V. Mashayekhi, and R. Rooholamini, "Architectural and performance evaluation of GigaNet and Myrinet interconnects on clusters of small-scale SMP servers," in *Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, (Washington, DC, USA), p. 18, IEEE Computer Society, 2000.
24. S. Majumder and S. Rixner, "Comparing Ethernet and Myrinet for MPI communication," in *Proceedings of the 7th workshop on languages, compilers, and run-time support for scalable systems*, (New York, NY, USA), pp. 1–7, ACM, 2004.
25. M. Lobosco, V. S. Costa, and C. L. de Amorim, "Performance evaluation of Fast Ethernet, Giganet, and Myrinet on a cluster," in *ICCS '02: Proceedings of the International Conference on Computational Science-Part I*, (London, UK), pp. 296–305, Springer-Verlag, 2002.
26. D. Turner, A. Oline, X. Chen, and T. Benjegerdes, "Integrating new capabilities into NetPIPE," in *Proceedings of the 10th European PVM/MPI Users' Group Meeting* (J. Dongarra, D. Laforenza, and S. Orlando, eds.), vol. 2840 of *Lecture Notes in Computer Science*, pp. 37–44, Springer, 2003.
27. R. Beck, A. Filion, P. Greidanus, G. Klok, C. Kuethe, P. Lu, C. Macdonell, A. Nisbet, and S. Portillo, "The Cluster Challenge: 6 Students, 26 Amps, 44 Hours," *Linux Clusters Institute*, 2008.
28. M. Roberts, A. Howe, and L. D. Whitley, "Modeling local search: A first step toward understanding hill-climbing search in oversubscribed scheduling," *International Conference on Automated Planning and Scheduling*, 2005.
29. H. Brunst, H.-C. Hoppe, W. E. Nagel, and M. Winkler, "Performance Optimization for Large Scale Computing: The Scalable VAMPIR Approach," in *ICCS '01: Proceedings of the International Conference on Computational Science*, pp. 751–760, Springer-Verlag, 2001.
30. "3.0 GHz Intel Core Duo, Intel compilers, 64-bit mode," 2008. `www.fftw.org/speed/CoreDuo-3.0GHz-icc64`.

31. C. A. Stewart, M. Link, D. S. McCaulay, D. Hancock, M. Pierce, G. Turner, R. Repasky, G. Rodgers, R. Aiken, D. Hancock, F. Saied, M. Mueller, M. Jurenz, and M. Lieber, "Implementation, performance, and science results from a 20.4 TFLOPS IBM BladeCenter cluster," *International Supercomputer Conference*, 2007.
32. Gordon Research Group, "GAMESS," 2008. `www.msg.ameslab.gov/gamess/`.
33. Environmental Molecular Sciences Laboratory, "TCGMSG Message Passing Library," 2004. `www.emsl.pnl.gov/docs/parsoft/tcgmsg/tcgmsg.html`.
34. R. M. Olson, M. W. Schmidt, M. S. Gordon, and A. P. Rendell, "Enabling the Efficient Use of SMP Clusters: The GAMESS/DDI Model," in *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, (Washington, DC, USA), p. 41, IEEE Computer Society, 2003.
35. Environmental Molecular Sciences Laboratory, "ARMCI—Aggregate Remote Memory Copy Interface," 2008. `www.emsl.pnl.gov/docs/parsoft/armci`.
36. B. J. Lynch, "Running Computational Chemistry Codes Efficiently," 2005. `www.msi.umn.edu/tutorial/chemistryphysics`.
37. H. Weller, H.G.; Tabor G.; Jasak and C. Fureby, "A Tensorial Approach to CFD using Object Oriented Techniques," *Computers in Physics*, pp. 620–631, Dec. 1998.
38. Center for Information Services and High Performance Computing, Dresden, Germany, *VampirTrace 5.6 User Manual*, 2008.
39. R. Schoene, G. Juckeland, W. E. Nagel, S. Pflueger, and R. Wloch, "Performance Comparison and Optimization: Case Studies using BenchIT.," in *PARCO* (G. R. Joubert, W. E. Nagel, F. J. Peters, O. G. Plata, P. Tirado, and E. L. Zapata, eds.), vol. 33 of *John von Neumann Institute for Computing Series*, pp. 877–884, Central Institute for Applied Mathematics, Juelich, Germany, 2005.
40. Intel Corporation, *Intel C++ Compiler User and Reference Guides*, 2008.
41. W. Wheeler, L. Aagesen, C. P. Arango, J. Faivovich, T. Grant, C. D'Haese, D. Janies, W. L. Smith, A. Varón, and G. Giribet, *Dynamic Homology and Phylogenetic Systematics: A Unified Approach Using POY*. American Museum of Natural History, New York., 2006.
42. Ros P. Stamatakis, Harald Meier, and Thomas Ludwig, "RAxML: A parallel program for phylogenetic tree inference,"