Parallel Supercomputing with Commodity Components

Michael S. Warren
Theoretical Astrophysics
Los Alamos National Laboratory
Los Alamos, NM 87545

Donald J. Becker
Goddard Space Flight Center
Code 930.5
Greenbelt, MD 20771

M. Patrick Goda
Theoretical Astrophysics
Los Alamos National Laboratory
Los Alamos, NM 87545

John K. Salmon
Center for Advanced Computing Research
California Institute of Technology
Mail Code 206-49
Pasadena, CA 91125

Thomas Sterling
Center for Advanced Computing Research
California Institute of Technology
Mail Code 158-79
Pasadena, CA 91125

Abstract

We have implemented a parallel computer architecture based entirely upon commodity personal computer components. Using 16 Intel Pentium Pro microprocessors and switched fast ethernet as a communication fabric, we have obtained sustained performance on scientific applications in excess of one Gigaflop. During one production astrophysics treecode simulation, we performed 1.2×10^{15} *floating point operations (1.2 Petaflops) over a three week period, with one phase of that simulation running continuously for two weeks without interruption. We report on a variety of disk, memory and network benchmarks. We also present results from the NAS parallel benchmark suite, which indicate that this architecture is competitive with current commercial architectures. In addition, we describe some software written to support efficient message passing, as well as a Linux device driver interface to the Pentium hardware performance monitoring registers.*

Keywords: Beowulf, treecode, benchmarks

1 Introduction

Moore's Law contends that the performance of a commodity microprocessor of a given price doubles every 18 months. Anyone involved with computer technology is well aware of the implications of this law. Current efforts in high-performance computing have been forced to rely on microprocessors and commodity DRAM to remain competitive, while adding value in areas such as network interconnects and proprietary operating system software and development tools.

The tremendous profit opportunity in hardware at the consumer level has driven the development of increasingly fast hardware accompanied by extreme price deflation. We have entered a phase where the different brands of CPU, disk storage, and network hardware have very little to distinguish them, apart from price. Building upon the foundation of the BEOWULF project [1], it has become possible to construct high-performance computers entirely out of commodity components and free software, thus obtaining a significant price/performance advantage over machines which require an expensive design and development phase. The development of powerful, free operating systems (Linux, FreeBSD), message passing standards (MPI), and support for a hardware independent peripheral interconnect bus (PCI), has finally allowed us to take advantage of the opportunity provided by low-cost commodity computer hardware.

The extreme disadvantage that all supercomputer vendors face is that they must be all things to all people, and yet ship a very limited volume of their high-priced product. If a customer requires a parallel debugger with a graphical user interface and an HPF compiler, it is a large and real expense to develop such software, which must be added to the cost of the hardware if the company intends to survive. In many cases, not enough of these costs were added, as is made clear by the list of bankruptcies and buy-outs in the supercomputer industry. Ideally, a self-sufficient user who does not require anything more than reliable hardware and an efficient send/receive message passing call should not be expected to pay for software which they will never use. This is, however, exactly the case with current parallel machines.

It is our view that many of the applications which require supercomputer performance will perform well on commodity parallel architectures. As one example, in 1992 two authors of this report were awarded a Gordon Bell Performance Prize [2] for "Astrophysical N-body Simulations Using Hierarchical Tree Data Structures" running on the 512 processor Intel Delta machine. It is now five years later, and it is possible to run that same simulation on a machine constructed out of mail-order parts and free software for a cost of less than $50,000. Not only that, we have found the commodity system to be as user-friendly and reliable as the current generation of parallel supercomputers.

Over the past three years, several important events have enabled commodity supercomputing:

- The dramatic increase in microprocessor performance has continued and accelerated. The Intel Pentium (P5) and Pentium Pro (P6) have improved floating point performance by more than a factor of ten over the i486. This has narrowed the performance gap between the high-volume Intel processors and the top-of-the-line workstation microprocessors significantly. The difference in raw floating point performance (on typical scientific workloads) between a 200 MHz Pentium Pro and the best DEC, SGI/Cray, HP/Convex, Sun or IBM has to offer is between a factor of three and five.

- A free Unix operating system (Linux [3]) has been written and is being used by hundreds of thousands of people. Originally designed for Intel family machines, releases are now running on Alpha, PowerPC and Sparc architectures. The freely distributable system has promoted development of high-quality device drivers for nearly every possible peripheral device (notably SCSI and Networking cards).

- Intel designed the Peripheral Component Interconnect (PCI) local bus [4] which offers greater than 100 Mbyte/second communication from the processor subsystem to the outside world, in a processor-independent manner. The PCI bus has become a de-facto standard, and allows peripheral cards (such as Fast Ethernet interfaces) to be plugged into any machine, regardless of CPU architecture.

- Fast Ethernet has leapfrogged other technologies such as ATM in terms of price/performance. 100 Mbit ethernet is capable of a bandwidth of over 10 Mbytes/sec between two points, with interface cards which cost less than $100 each. Commodity switches are now available which allow up to 16 machines to communicate with each other at the full 100 Mbit bandwidth, at a cost of a few hundred dollars per port.

- The MPI standard has enabled the development of a reasonable amount of portable parallel software. Groups working on machines such as the CM-5, Intel Paragon, IBM SP-2, and Cray T3D have spent considerable effort over the past several years to develop such software, and can take immediate advantage of new machines which support the MPI message passing standard and a suitable UNIX software environment.

2 Architecture

We have constructed two distinct 16 processor Pentium Pro machines, each having 2 Gbytes of RAM, and either 50 or 80 Gbytes of disk space. The machines differ primarily in their network topology. The cost of the machines in the fall of 1996 was roughly $50,000—$60,000. The price to construct an equivalent system as we go to press in May 1997 is less than $40,000.

2.1 Loki and Hyglac

At the Theoretical Division of Los Alamos National Laboratory, Loki [5] was constructed from 16 nodes as described in Table 1. The whole machine contains 2 Gbytes of memory and 50 Gbytes of disk. The four-port full-duplex ethernet cards allowed us to experiment with a variety of network topologies, such as a hypercube. The fifth ethernet port was connected to one of two 3Com SuperStack II Switch 3000 TX 8-port full-duplex Fast Ethernet switches, which provide connectivity to a front-end server, as well as bypassing the multihop hypercube routes. An early lesson we learned is that the memory bandwidth of the Pentium Pro Natoma chipset is not sufficient to support more than about 20 Mbytes/sec of message traffic per node when using TCP or UDP protocols (due to copies of data from the kernel to user space), thus for the results quoted in this paper, Loki was connected in a split-switch topology, using only two ethernet ports per node. The video card in each node is not strictly necessary, but is required to access the initial BIOS setup screen, and to see certain hardware error messages.

At Caltech/JPL, Hyglac [6] was constructed from 16 nodes which were almost identical to those of Loki. The primary differences were the use of D-Link DFE-500TX 100 Mb Fast Ethernet Cards ($85 each), a Bay Networks 28115 16-way Fast Ethernet Switch, two Western Digital 2.52 Gbyte drives per node, and the use of EDO DRAM. The total price of Hyglac (including 8.75% sales tax) was $50,498.

2.2 Advantages of a Commodity Architecture

Commodity hardware can be upgraded with little difficulty. The machines described here could be upgraded to a different brand of microprocessor, such as a 500 Mhz DEC Alpha CPU, by replacing only the motherboard and processor. The memory, disk, and network systems can be re-used. Also, since the Linux operating system supports both the Pentium and Alpha, the software environment would remain the same through such an upgrade.

Maintenance contracts for current workstations and supercomputers are expensive. One can expect to pay 5-10% of the price of the machine per year in hardware support costs. Hardware support is much less demanding for commodity parallel architectures, which consist of many replicated parts. It is a simple matter to keep spare parts (or a spare node) on hand. This converts the usual ordeal of emergency phone calls to the vendor into a ten minute swap-out of the offending part. Many parts (memory and network cards) are traditionally covered by a lifetime warranty, and no single part of the machines described here costs more than $700 to replace (apart from the ethernet switch). Even more significantly, the software maintenance of these systems is significantly less complex than other machines of equivalent performance, which is a significant advantage for many small groups which would not be able to afford the additional system management expense of a typical parallel supercomputer.

The technology described here is not complex. In fact, both machines described in this paper were sent in shipping crates to Supercomputing '96 in Pittsburgh in November 1996, where they were connected with some additional ethernet hardware to form a single 32 processor machine which was demonstrated during the conference.

3 Performance Results

3.1 Low-level benchmarks

We present measures of disk and memory bandwidth in Table 2 using the STREAM [7] and lmbench [8] performance analysis tools. We note that the memory bandwidth improvement coming from EDO DRAM is clearly discernible, with EDO improving memory read bandwidth using the Intel VS440FX board by 40%, and write and copy bandwidth to a somewhat lesser degree. Although we do not have space to present the detailed data, we

Table 1: Loki architecture and price (September, 1996).

Qty.	Price	Ext.	Description
16	595	9520	Intel Pentium Pro 200 Mhz CPU/256k cache
16	15	240	Heat Sink and Fan
16	295	4720	Intel VS440FX (Venus) motherboard
64	235	15040	8x36 60ns parity FPM SIMMS (128 Mb per node)
16	359	5744	Quantum Fireball 3240 Mbyte IDE Hard Drive
16	657	10512	Cogent EM400 TX PCI Quartet Fast Ethernet
16	129	2064	SMC EtherPower 10/100 Fast Ethernet PCI Card
16	59	944	S3 Trio-64 1Mb PCI Video Card
16	119	1904	ATX Case
2	4794	9588	3Com SuperStack II Switch 3000, 8-port Fast Ethernet
		255	Ethernet cables
Total		$60,531	

Table 2: Memory and Disk micro benchmarks. Bandwidth is reported in Mbytes/sec. Identical Loki nodes were used to compare Fast Page Mode (FPM) Dynamic Random Access Memory (DRAM), and Extended Data Out (EDO) DRAM.

Benchmark	BW (FPM)	BW (EDO)
Stream Copy	80	100
Stream Scale	80	100
Stream Add	88	109
Stream Triad	88	107
lmbench Mem Read	164	231
lmbench Mem Write	65	83
lmbench Mem Copy	46	53
Disk R/W	4.2	

have found that EDO DRAM typically improves performance on the NAS benchmarks (which are representative of many Fortran codes) by about 20%. On the SPEC95 benchmark suite, EDO improves performance to a lesser degree (about 8% on average, but three of the components show increases of 15%).

In Table 3 we present network bandwidth and latency numbers for a variety of protocols. The Salmon-Warren Message Passing Interface (SWAMPI) is a small and fast implementation of a minimal set of Message Passing Interface (MPI) functions based on TCP sockets, and is described later in this paper. MPICH [9] is the portable MPI implementation from Argonne National Laboratory and Mississippi State University. LAM [10] is the portable MPI implementation from Ohio State University. U-Net [11] is a User-Space message library, which avoids the UDP or TCP socket overhead that the other protocols incur.

Table 3: Comparison of various message passing protocols. SWAMPI, MPICH 1.0.13, TCP, UDP and U-Net numbers are with Linux 2.1.29, others with 2.0.29. Bandwidth is reported in Mbytes/sec. Latencies are *round-trip* time in microseconds. The last column includes the latency added by the fast ethernet switch (as opposed to a direct connection with a crossed RX/TX cable).

Version	BW	Lat.	Lat. w/sw
SWAMPI	11.7	208	238
MPICH 1.0.13	3.2		503
MPICH 1.1.0	8.8		390
LAM 6.1	7.3		2690
LAM 6.1 -c2c	11.6		322
TCP socket	11.7	158	182
UDP socket	11.7	131	153
U-Net	12.3	55	

3.2 Application Benchmarks and Results

3.2.1 The Hashed Oct-tree code

Two of the authors of this paper (Warren & Salmon) have developed a considerable software infrastructure to support large N-body simulations [12, 13]. Needing little more than a function to send and receive messages, this code has been tuned and refined over several generations of parallel architectures since being awarded a Gordon Bell Prize in the performance category in 1992 [2]. It offers one reasonable way to compare the commodity parallel architecture with other parallel supercomputers, since we have made a concerted effort to optimize the code for all of these architectures.

The basic algorithm may be divided into several stages. Our discussion here is necessarily brief. For a more detailed discussion of the intricacies of this code, please consult the references or our Web page (`http://qso.lanl.gov`). First, particles are domain decomposed into spatial groups. Second, a distributed tree data structure is constructed. In the main stage of the algorithm, this tree is traversed independently in each processor, with requests for non-local data being generated as needed. In our implementation, we assign a `Key` to each particle, which is based on a self-similar space-filling curve (Morton ordering). This maps the points in 3-dimensional space to a 1-dimensional list, which maintains as much spatial locality as possible. The domain decomposition is obtained by splitting this list into N_p (number of processors) pieces. The implementation of the domain decomposition is practically identical to a parallel sorting algorithm, with the modification that the amount of data that ends up in each processor is weighted by the work associated with each item.

The Morton ordered key labeling scheme implicitly defines the topology of the tree, and makes it possible to easily compute the key of a parent, daughter, or boundary cell for a given key. A hash table is used in order to translate the key into a pointer to the location where the cell data are stored. This level of indirection through a hash table can also be used to catch accesses to non-local data, and allows us to request and receive data from other processors using the global key name space. An efficient mechanism for latency hiding in the tree traversal phase of the algorithm is critical.

All of this data structure manipulation is to support the fundamental approximation employed by treecodes:

$$\sum_j \frac{G m_j \vec{d}_{ij}}{|d_{ij}|^3} \approx \frac{G M \vec{d}_{i,cm}}{d_{i,cm}^3} + \ldots, \qquad (1)$$

where $\vec{d}_{i,cm} = \vec{x}_i - \vec{x}_{cm}$ is the vector from $\vec{x}_i$ to the center-of-mass of the particles that appear under the summation on the left-hand side, and the ellipsis indicates quadrupole, octopole, and further terms in the multipole expansion.

For the treecode benchmark that follows, all machines were running the same `C` code, with the exception of the machine-specific optimizations to the inner loop. The Intel i860 machines and the CM-5 have the inner loop coded in assembly language. Several of the machines (SP-2, T3D and Loki) decompose the reciprocal square root function required for a gravitational interaction into a table lookup, Chebychev polynomial interpolation, and Newton-Raphson iteration, using the algorithm of Karp [14]. This algorithm uses only adds and multiplies, and requires 38 floating point operations per interaction, while we count 28 floating point operations per interaction for the machines which use a hardware square root function. Since this skews comparison of the actual application performance, we also quote `Mnewtons`, which are the number of gravitational interactions performed per second.

3.2.2 A 1.2 Petaflop simulation

Between April 18 and May 8 1997, we ran a simulation with 9.7 million particles on the 16 processors of Loki for 1047 timesteps. The simulation was interrupted due to a scheduled power outage in our machine room on April 25. Between April 25 and May 8, the code ran continuously for 13.5 days, with no restarts. For those of us in the habit of staying up all night watching our job run on the latest hardware at the supercomputer center (so we can restart it when it crashes, because our allocated

Table 4: Treecode performance results. We calculate the forces on 10 million particles to an accuracy of 0.1% (except for the first line, where we use 322 million particles). The times reported are wall-clock time in seconds, and they includes all message passing and load imbalance overheads. The 10 Mparticle benchmark requires the computation of 70.668 billion gravitational interactions. Some new inner loop optimizations were applied to the final result in the table (Loki), so attempting to extrapolate the scaling efficiency for Loki+Hyglac is not possible.

Site	Machine	Procs	N $\times 10^6$	Time	Mnewtons	Gflops	Mflops/proc
Sandia	ASCI Red	6800	322	98.4	12180	464.9	68.4
Sandia	ASCI Red	4096	10	16.4	4322	164.3	40.1
LANL	TMC CM-5	512		140.7	502	14.06	27.5
Caltech	Intel Paragon	512		144.4	489	13.70	26.8
NRL	TMC CM-5E	256		171.0	413	11.57	45.2
Caltech	Intel Delta	512		199.3	355	10.02	19.6
NAS	IBM SP-2 (66/WN)	128		281.9	251	9.52	74.4
JPL	Cray T3D	256		338.0	209	7.94	31.0
LANL	SGI Origin 2000	24		394.2	179	5.02	209.0
LANL	CM-5 no vu	256		754.6	94	2.62	5.1
SC '96	Loki+Hyglac	32		1218	58	2.19	68.4
LANL	Loki	16		2102	34	1.28	80.0

time is too valuable to lose) running a job that size without an interruption is nearly miraculous.

The simulation was of a spherical region of space 100 Mpc (Megaparsec) in diameter; a region large enough to contain a few hundred thousand typical galaxies. The region inside a sphere of diameter 100 Mpc was calculated at high mass resolution, while a buffer region of 50 Mpc with a particle mass 8 times higher was used around the outside to provide boundary conditions. The initial conditions were extracted from a 134 million point initial dataset, calculated using a a 512^3 point 3-d FFT on Loki, from a Cold Dark Matter power spectrum of density fluctuations. Overall, the simulation carried out 1.2×10^{15} floating point operations (1.2 Petaflops). We created 40 data files totaling over 13 Gbytes. A single data file from this simulation is 312 Mbytes is size, and they were written striped over the 16 disks in the system, obtaining an aggregate I/O bandwidth of well over 50 Mbytes/sec. The entire simulation required the computation of 3.16×10^{13} interactions during a wall clock time of 1,474,000 seconds (410 hours, just over seventeen days), for an overall throughput of 815×10^6 floating point operations per second (815 Mflops). This simulation on Loki consisted of as many operations as any single simulation we had performed on any parallel supercomputer prior to April 1997 (when we performed a simulation on the ASCI Intel Teraflops system with 32 times as many particles, and 8 times as many floating point operations).

A better performance result was obtained during the initial 30 timesteps of the same simulation. We computed 1.15×10^{12} interactions in 36973 seconds of wall-clock time, for an overall throughput of 1.186×10^9 floating point operations per second (1.19 Gflops). This result is better than the 815 Mflops quoted above since the initial stages of the calculation require less time spent in tree traversal overhead. It is important to note that much of the useful work accomplished by the treecode algorithm has nothing to do with floating point operations, and is not reflected in the number of Mflops which we report. The only purpose of using a treecode is to *avoid doing* as many floating point operations as possible.

3.2.3 The NAS Parallel benchmarks

The results shown in Table 5 use the NAS Parallel benchmarks version 2 [15], and results for the SGI were obtained from the NPB Web page [16]. On the Pentium Pro, one must take great care to assure that the compiler flags and operating system provide 8-byte alignment for all double precision floating point variables, since performance can suffer greatly if proper alignment is not maintained. This is especially true with `g77`, where the `-malign-double` flag can improve performance by a factor of two or more. One must also be sure to use Linux `libc` version 5.4.23 or greater to assure that the stack is properly aligned.

Table 5: Sixteen processor performance (Mops) for Class B NPB 2.2 benchmarks. Data from Loki with the Portland Group (PGI) pgf77 Rel 1.3-3 Fortran 77 compiler, the GNU 2.7.2.f.1 compilers, and the ASCI Intel Teraflops system with PGI Rel 1.3-4a, and an SGI Origin 2000 are presented.

	Loki		ASCI	SGI
	PGI	GNU	Red	Origin
BT	354.6	331.4	445.5	925.5
SP	255.5	224.5	334.8	957.0
LU	428.6	403.7	490.2	1317.4
MG	296.8	267.1	363.7	1039.6
FT	177.8			648.2
EP	8.9	12.7	7.1	68.7
IS	14.8	14.6	38.0	33.9

Table 6: Data for the NPB 2.2 Class A benchmarks on Loki. The first column denotes the number of processors, and the data are Mflops/sec (Mops/sec for IS). This data is plotted in Figure 1.

NC	BT	SP	LU	FT	MG	IS
1		19	31			2.5
2			59			3.2
4	94	71	118	73	78	5.7
8			222	134	161	9.3
9	202	122				
16	358	242	453	250	281	15.0

4 Tools in support of Parallel Computation

4.1 Timing and Performance Monitoring

The Pentium architecture supports a 64-bit counter which is incremented every clock tick, and requires only a few assembly instructions to access [17]. On a 200 MHz Pentium Pro this supplies a real-time clock with a 5 nanosecond tick size, which is invaluable for performance tuning and providing messages with accurate timestamps.

Pentium and Pentium Pro processors also support an advanced set of hardware diagnostic timing and counting routines. These allow direct measurement of very small time intervals, as well as counting cache misses and other profiling support. Users access these capabilities through `perfmon`, a program we designed which allows access to the performance monitoring features of the Pentium and Pentium Pro. Both of these processors allow monitoring of two performance events simultaneously. On the Pentium Pro there are 68 events which can be monitored, including floating point operations, floating point divides or multiplies, memory references, L2 cache loads or stores and outstanding bus requests.

To facilitate monitoring of more than two events over the life of a process, we developed a complement to `perfmon`. `mperfmon` multiplexes over all performance monitoring events (or some subset therein) to produce an ensemble approximation of processor utilization. For codes which run for longer than about 5 sec (on the Pentium Pro), the user receives an accurate estimate for all of the performance monitoring events.

`perfmon` and `mperfmon` allow a glimpse into the internal workings of the Pentium and Pentium Pro that has not previously been possible under most operating systems. These tools can be used on production code or code fragments to provide insight into code performance. For the astute programmer these performance monitoring tools often provide sufficient information to determine if certain optimizations are warranted, and exactly how

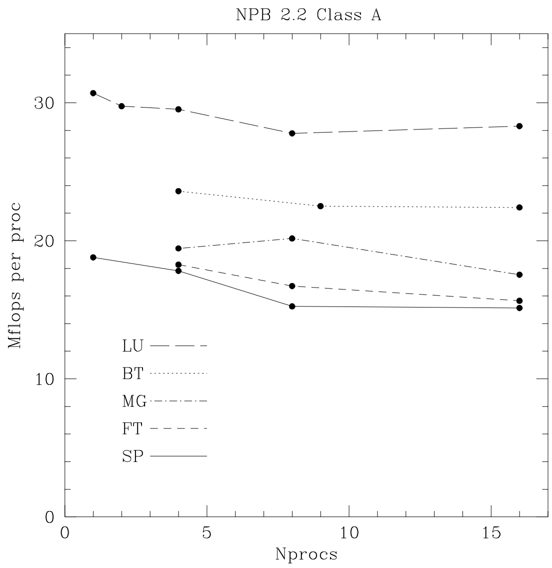

Figure 1: Scaling of the NAS Class A benchmarks on Loki.

they might increase performance.

4.2 Message Passing Interface (MPI)

There are several publicly available implementations of the Message Passing Interface (MPI) library. While it is possible to perform computations using these libraries, their performance is not optimal. This is certainly not the fault of the implementation; it is simply impossible to support maximally efficient communication using a maximally portable library. After close inspection of the available MPI codes, we decided to write our own minimal implementation from scratch, rather than wade through the layers upon layers in the MPICH and LAM codes. Rather than spend extensive effort implementing over 120 MPI functions, we concentrated on efficient implementations of the basic functions: `isend, irecv, test, wait` and the important collective functions: `reduce, bcast, and gather.` After roughly one week of effort, we had a minimal MPI library capable of running our treecode, as well as all of the version 2 MPI implementation of the NAS parallel benchmarks.

This minimal library implements the most important MPI functions for both `C` and `Fortran` in a few thousand lines of source and header files (including comments). This may be compared to about 100,000 lines of code in the Ohio State LAM implementation, and 250,000 in the MPICH distribution (of which 40k lines are examples, and 100k lines are device specific). Needless to say, it is considerably easier to understand what is happening in 2000 lines of our own code vs 100,000 of somebody else's. Currently, the performance of our own MPI (which we have provisionally named SWAMPI, for the Salmon-Warren Message Passing Interface) has been measured to improve message bandwidth significantly over the general purpose implementations. Also, while the performance of LAM with the `-c2c` flag (which forces the use of a more efficient protocol at the expense of other features) is quite close to that of SWAMPI, we found that LAM with `-c2c` was unable to run many programs that ran fine without the `-c2c` flag.

5 Discussion

It is clear that commodity parallel processing (CPP) architectures such as Loki and Hyglac can perform in the role of "departmental supercomputer." The most convincing evidence we put forth is the fact that we can (and have) performed simulations requiring 2 Gbytes of memory and 10^{15} operations. The overall flop rates we measure on the NPB version 2 Class B benchmarks (240-360 Mflops) are about a factor of two less than a single CPU Cray C90 running the NPB version 1 benchmarks. (The version 1 NPB benchmarks have no restrictions on how one optimizes the algorithms, which are defined in "pencil and paper" fashion, so if someone were willing to spend the time to optimize the version 1 benchmarks for Loki, the advantage of the C90 would be even less.) The main advantage of CPP architectures may be that they are affordable enough to be used as dedicated computing resources, achieving an overall throughput equivalent to larger machines which must be shared with many other users.

5.1 Limitations

As presented here, the CPP architecture is unlikely to scale up very well. Although Loki has been configured with 5 fast ethernet interfaces, which should be capable of an aggregate 60 Mbytes/sec, we have found in practice that one can drive no more than two fast ethernet interfaces to their full potential. It seems the memory bandwidth on a Pentium Pro motherboard with the Natoma chipset is not capable of keeping up with more than two fast ethernet ports at a time (when conventional TCP or UDP message protocols are used). This limits overall performance into or out of a node to roughly 20 Mbytes/second, while we have shown that the PCI bus on this platform is capable of supporting rates of at least 40 Mbytes/sec. Using faster network hardware would not help. In order to obtain better performance, one needs greatly increased memory copy bandwidth, or else an implementation of active messages or U-Net to avoid *all* buffer copies between kernel space and user space.

5.2 Comparing machines

While all the benchmarks reported in this paper were run on Loki, a similar suite of benchmarks were run on Hyglac, and the results were similar within the limitations that the two machines were never running quite the same versions of the operating system software or compilers, and that Hyglac used higher-bandwidth EDO memory instead of FPM memory. We can also compare Loki to the results obtained on 16 nodes of the ASCI Red system (Janus, which incidentally is binary compatible with Loki at the object file level). Janus has exactly the same Pentium Pro processor, amount of memory, and compiler as Loki. The differences were the network on Janus is about 15 times faster (160 Mbytes/sec), the latency is less (60 microseconds round-trip), and the memory bandwidth is higher. Overall Janus has an advantage at the 16 processor level that ranges from 10%-30% (Table 5). We estimate that roughly half of this improvement comes from the better memory bandwidth of the Janus nodes. Thus, we conclude (surprisingly!) that using switched fast ethernet instead of an exotic networking technology appears to have a fairly small effect on performance for a 16 processor machine on the NAS benchmarks (with the exception of the message bandwidth hungry IS benchmark). The effect of improved memory bandwidth from the interleaved memory on Janus is pratically the same as its network advantage. Unfortunately, Intel has discontinued the Orion chipset which supports interleaved memory, and we must wait patiently to see what the new generation of Intel chipsets with SDRAM have to offer. We have seen that memory bandwidth is critical to both floating point performance, as well as network performance. It is for this reason that using multiple processors within each node (SMP) is unlikely to be a good idea for many applications. It simply makes the shortage of memory bandwidth even worse.

In terms of price/performance, one might note that at the present time (May 1997), just buying 2 Gbytes of memory for an SGI Origin 2000 machine costs roughly twice as much as the whole of Loki or Hyglac. While one would be in the neighborhood of the price and floating point performance of Loki with a no-frills 4-CPU Origin 200, the amount of memory and disk space in such an entry-level machine is certainly not capable of supporting large computations. One of the most striking features of Loki and Hyglac is how their excellent machine balance has fallen naturally out of spending roughly equivalent amounts on the processor, memory, disk and network subsystems. One should be careful not to fall into the trap of buying a machine loaded with horsepower, but lacking the memory and I/O subsystems which are absolutely required to accomplish any useful work.

Taking a look at the vendor reported prices in Nov. 1996 for NPB Class B capable machines as reported in [18] ($960,000 for a 24 processor Origin 2000, $3,520,000 for a 64 processor IBM SP-2 P2SC, and $580,000 for a DEC AlphaServer 8400 5/440), we find that the price/performance of our CPP machines on the NPB 2.2 MPI/Fortran 77 benchmarks are better by a factor of three or more. For example, the time for a 64 processor SP-2 P2SC-120 machine to run the NPB version 2 Class B BT benchmark is 118 seconds, beating Loki in speed by a factor of 17, at a cost about 60 times higher. The SGI runs the benchmark in 471 seconds, 4.2 times faster, at a cost 16 times higher.

6 Conclusion

Commodity components offer an alternative route to follow in search of supercomputer performance. Beyond the mere cost advantage (which can be nearly an order of magnitude in some cases) there are many intangible benefits such as flexibility, upgradability, and the ability to have control of your own computing resources. We have shown that off-the-shelf technology can now compete successfully with the best commercial offerings for some problems. At present, it may be arguable whether commodity supercomputing is genuinely superior, but the advantages inherent in mass-market technology will only continue to grow.

References

[1] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 1995 International Conference on Parallel Processing (ICPP)*, pages 11–14, 1995.

[2] M. S. Warren and J. K. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Supercomputing '92*, pages 570–576, Los Alamitos, 1992. IEEE Comp. Soc.

[3] The Linux documentation project. http://sunsite.unc.edu/mdw/linux.html.

[4] T. Shanely and D. Anderson. *PCI System Architecture*. MindShare, Inc., Richardson, TX, 1992.

[5] M. S. Warren and M. P. Goda. Loki – commodity parallel processing. http://loki-www.lanl.gov.

[6] J. Lindheim, J. K. Salmon, and T. Sterling. Cacr's beowulf machine. http://www.cacr.caltech.edu/research/Beowulf.

[7] John McCalpin. Sustainable memory bandwidth in current high performance computers. http://www.cs.virginia.edu/stream/ref.html.

[8] Larry McVoy. lmbench: Portable tools for performance analysis. http://reality.sgi.com/lm_engr/lmbench.

[9] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. MPICH — a portable implementation of MPI. http://www.mcs.anl.gov/Projects/mpi/mpich.

[10] Ohio State University. LAM/MPI parallel computing. http://www.osc.edu/lam.html.

[11] Matt Welsh, Anindya Basu, and Thorsten von Eicken. ATM and fast ethernet network interfaces for user-level communication. In *Proceedings of the Third International Symposium on High Performance Computer Architecture (HPCA), San Antonio, Texas*, 1997.

[12] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Supercomputing '93*, pages 12–21, Los Alamitos, 1993. IEEE Comp. Soc.

[13] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87:266–290, 1995.

[14] A. H. Karp. Speeding up N-body calculations on machines without hardware square root. techical report, 1992.

[15] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0. Technical report, NASA Ames Research Center, 1995.

[16] NPB 2 detailed results: Graphs and raw data 04/21/97. http://science.nas.nasa.gov/Software/NPB.

[17] Intel Corporation. *Pentium Pro Processor Developer's Manual*, volume 2. McGraw-Hill, 1996.

[18] Subhash Saini and David H. Bailey. NAS parallel benchmark (version 1.0) results. Technical report, NASA Ames Research Center, 1996.