Experiences with Legion on the Centurion Cluster*

Greg Lindahl   Steve J. Chapin
Norman Beekwilder   Andrew Grimshaw
The Legion Project
Department of Computer Science
University of Virginia
Charlottesville, VA 22903–2442
{lindahl,chapin,nfb5z,grimshaw}@cs.virginia.edu

DRAFT

Abstract

The Legion Project spends most of its time building metacomputing environments and applications, but we are also actively building a clustered system, which we have named Centurion. Centurion is currently a cluster of 64 Alpha PCs, networked with low-latency gigabit networking hardware from Myricom, joined by a dozen x86-architecture PCs. We plan on doubling the size of the cluster in the near future. This shared-nothing, heterogeneous environment is an ideal fit for the Legion metacomputing system, which provides services such as naming, scheduling, heterogeneous communications, and parallel I/O. Centurion is roughly the size of "small" HPC systems in use today at super-computing centers, giving us the opportunity to compare price and performance with traditional systems, and investigate the hardware and software challenges of building large clusters with commodity hardware.

This paper will cover our experiences with Centurion and Legion. We begin with an overview of the Legion metacomputing system. We will then discuss the costs and hidden costs of assembling a cluster. We then describe the performance of Centurion, using microbenchmarks of the communications system, standard parallel application benchmarks, and user applications. User applications running on the system come from a variety of scientific disciplines, and range from traditional MPI codes taken straight from supercomputers to more novel applications using "bag of tasks" and macro-dataflow formalisms. Within this range of applications, we will discuss our successes and failures with hiding network latency, load balancing, and most importantly, usability of the system by researchers without extensive training in parallel computing.

Keywords: cluster computing, heterogeneous computing

1

1   Introduction

The coming of gigabit networking technology makes possible the construction of nationwide virtual computers comprising workstations, high-performance supercomputers, and clusters of machines. To realize the potential that the physical infrastructure provides, software must be developed that is easy to use, supports large degrees of parallelism in applications code, and manages the complexity of the underlying physical system for the user. The Legion project at the University of Virginia is developing such software [5]. Legion addresses issues such as parallelism, fault tolerance, security, autonomy, heterogeneity, resource management, and access transparency in a multi-language environment.

As part of this effort, we are building our own clustered system, called Centurion. Centurion is currently composed of 64 Digital Equipment Corporation Alpha processors, and will soon be expanded to be over 200 machines and 300 processors. These processors are interconnected by both a high-speed, low-latency system-area network intended for heavy-duty application communication (Myrinet), and a partially-switched 100 Mbps Ethernet control network.

We have learned much in the process of constructing both the hardware and software for this system. Our experiences range from basic hardware assembly through system software configuration and run-time system development to application deployment. In this paper, we will describe some of our experiences in each of these areas. In particular, the ultimate yardstick for success is whether our system can successfully run scientific applications in a cost-effective manner.

In the remainder of this paper, we will give overviews of the run-time system and the hardware, describe our experiences with the hardware, discuss related work, provide performance measurements, describe applications, and give conclusions.

2   The Software: Legion

Legion is an object-oriented distributed computing system. The Legion design encompasses ten basic objectives: site autonomy, support for heterogeneity, extensibility, ease-of-use, parallel processing to achieve performance, fault tolerance, scalability, security, multi-language support, and global naming. These objectives are described in greater depth in Grimshaw et al. [5]. Many of the features supplied by Legion are superfluous in the environment under discussion—support for heterogeneity, parallel processing, and ease-of-use are of primary interest to us here. The other features are important in the context of connecting clusters from remote sites.

Supporting heterogeneity requires Legion to accommodate vastly differing computing capabilities among constituent machines, including differences in architectures, operating systems, and installed software. Such support is important to run complex distributed computations, such as a weather forecasting and visualization program—portions of the computation may be best suited for vector supercomputers, message-passing architectures, or graphics workstations.

In the context of Centurion, Legion provides automatic binary management to ensure that the same program can run on multiple architectures simultaneously. Legion's automatic binary management is one of the ways it simplifies use of a distributed system. Other salient features include a unified namespace to provide location-transparent execution and support for automatic checkpointing, migration, and fault recovery.

Legion's programming model, the macro-dataflow model, supports both intra-object and inter-object parallelism. The macro-dataflow model is based on dataflow graphs, where arcs represent communication and nodes represent computation. The dataflow model is realized in Legion because an object waits for all incoming data items and then automatically invokes the requisite method in the dataflow graph. Results of the method invocation are passed along the arcs in the macro-dataflow graph (which may imply returning results to the caller, or passing them to one or more separate objects in Legion space).

Legion also support fault tolerance, which is a boon to cluster builders. Because objects in Legion can be transparently migrated, failure of a single node does not negatively affect the rest of the system. This graceful degradation at the software level is necessary in an environment comprised of cheap, interchangeable hardware, and facilitates hot-swapping of failed components.

3 The Hardware: Centurion

3.1 Phase I

The original 64 nodes of Centurion were funded by an NSF Major Research Infrastructure grant. This grant recognized that computation is increasingly central to the conduct of science and engineering, augmenting both theory and experimentation. The purpose of this grant was to develop a platform for collaboration between computer scientists and discipline scientists (biologists, material scientists, physicists, etc.).

The original suite of target applications included those from biochemistry, chemical engineering, computer science, electrical engineering, engineering physics, environmental science, and materials science. All of the applications shared a need for significantly more computation power than had been available. Further, all of the applications were either already parallelized or are being parallelized in collaboration with the computer science team members.

Thus, we set out to build a 25 gigaflop multicomputer from commercial, off-the-shelf (COTS) components, with the express purpose of supporting scientific computation. We ended up constructing a 60+ peak gigaflop COTS machine. In the process, we faced many tradeoffs and learned quite a bit about cluster construction.

3.2 Hardware tradeoffs

The usual problem of buying any system is to spend a fixed pot of money to buy the fastest system for whatever problem one plans to run. Like most computer scientists, we had a huge variety of problems in mind, so we used the SPEC95 cpu benchmarks to assess

individual node performance. We were happy to consider any CPU, but there is no cost-effective way to buy SPARC or MIPS chips, and we were ignorant about cheap Power PC options. This left x86 and alpha as options. The next question revolved around the OS. We knew how to cheaply administer Linux; however, Linux doesn't run the compilers usually used to generate SPEC95 results. Using some benchmarking results for a few combinations of hardware, Linux, and gcc/g77, we convinced ourselves that there was approximately a 10% to 20% performance loss on both the x86 and Alpha systems in question due to using gcc/g77 verses the best commercial compiler, and that gcc/g77 (mainly egcs) was steadily improving these results. So we felt confident in using SPEC95 with proprietary compilers to compare hardware, with the expectation that the relative performance with gcc/g77 would be the same, even if the absolute performance were somewhat worse.

At the time of our benchmarking, the SPEC95 integer and floating point benchmarks for the most cost-effective Alpha-based system were roughly twice that of a single x86-based system, and dual x86 systems were roughly the same cost as single-cpu Alpha systems. We chose single-cpu Alpha systems. We return to the compiler question later in this paper. In addition, we had 16 dual-processor Intel Pentium-Pro-based machines acquired piecemeal from various sources.[1]

We wanted to buy "a lot" of network bandwidth because we wanted to attack traditional supercomputing problems, which commonly require large bandwidth and low latency communications. ATM was extremely expensive and not that much faster than 100 megabit switched Ethernet. Gigabit Ethernet was incapable of connecting large numbers of nodes at the time of our purchase. The only moderately expensive, high-bandwidth, low-latency network available at that time was Myrinet. We paid approximately $3,000 per node for the cpu, memory, and disk; Myrinet cost $1,700 per node for a low-latency network with a bisection bandwidth of 32 gigabit/sec. for a 64 node system. While spending 40% of our total money on networking may seem extravagant, we expect that networks with this level of performance to be much cheaper next year, and we wanted to experiment with next year's typical ratio of communications to computation.

In addition to this fast and expensive network, we also spent a small amount of money per node for a "minimal" Ethernet network. The cheapest Ethernet topology that we could think of consisted of a 16-port 100 megabit switch connecting 9 100 megabit hubs, each with 7 systems. This cost approximately $110 per node, including cards and cables. In contrast, a 64-port 100 megabit switch costs around $400 per node.

3.3 Experiences with Centurion

The first question most people ask when they consider buying a cluster of individual machines instead of a single, large machine, is "aren't there a lot of hidden costs of ownership, ranging from assembly to higher maintenance costs?" These concerns are doubled when the cluster in question runs Linux and is not assembled by a vendor. We have attempted

---

[1]We refer to the collection as Stone Soup because of the similarity in our PC acquisition to the manner in which the soldier accumulated soup ingredients in the popular folk tale.

to assess these costs. Unfortunately (or fortunately?), our team includes a sysadmin who has commercial experience administering hundreds of machines, which may not be typical for most academic departments.

Initial physical assembly and installation of the OS via disk cloning took approximately 2 hours of semi-skilled labor per node; the cost of this labor is only a small fraction of the cost of a node. We later had to re-install a different OS version from scratch; for that we worked out a network install that took approximately 10 minutes of labor per node. We administer the machines using RedHat "packages" and a few simple iterators which copy files or perform commands on all nodes. We have not had the cluster for long enough to assess long-term maintenance costs; we did lose 2 cpus and 1 disk (out of 64) to infant mortality. We had prepared for this situation by ordering spare units which were kept ready to be swapped in. Again, the Legion run-time environment facilitates this kind of hot-swap repair at the node level.

We did suffer a bit because of our choice in operating systems. We chose Linux because we desired a free Unix clone that ran on both Alphas (our new architecture) and Intel boxes (our old architecture). Also, this helped us to remain compatible both with our research partners at the Department of Energy National Labs, and with earlier cluster efforts. Red-Hat 5.0/Alpha was shipped with broken C++ libraries, and RedHat was uninterested in fixing it, so we downgraded to 4.2. Driver support for Alpha Linux takes a back-seat to x86 drivers; we had to look around a bit to find out which driver versions actually worked with Alpha Linux. We discovered that we could crash all 64 machines simultaneously with certain floating point exceptions; that was solved by installing a newer kernel version (2.0.33). Myrinet's network software did not work correctly with a network this large; other large Myrinet networks do not use the drivers supplied by Myricom. However, Myricom was able to fix their driver in 2 days.

Thus, the operating system was a case of getting what we paid for, and we complicated the matter by choosing the highest-performance architecture available, not the one for which the OS had the best support. We knew this was the case when we made the decision to go with Linux, and we are, overall, satisfied with the outcome. We are considering a move to Digital Unix on the Alpha boxes, but this would bring its own set of challenges (compiler support, OS incompatibility with our nodes and our research collaborators, etc.).

3.4 Phase II

For Phase I of Centurion, we purchased a homogeneous machine with relatively good (and expensive) networking. For Phase II, which is funded by the Office of Naval Research through the DURIP program, we want to experiment with scheduling heterogeneous processors and networks, so we will be buying a large number of x86 and alpha processors, and use a large 100 megabit Ethernet switch to link them.

The machine will then have several degrees of heterogeneity, including processor type, memory per node, interconnection network, and possibly system software. The purpose of the DURIP award is to study application performance and scheduling in a heterogeneous environment, so that we can develop technology usable by the DoD to best run their com-

plex applications. Phase II will provide us with a rich environment for operating systems research, as well.

4   Related Work

There are three other projects which we believe are of primary importance when discussing cluster work such as Centurion. First, the Beowulf project [7] at NASA is one of the best-known "pile of PC" cluster projects. However, Beowulf systems aim for the lowest-price point rather than the best price-performance tradeoff. This has advantages in that there is less risk—free operating system support is better for Intel platforms than for others, and well-understood technologies such as Ethernet provide no surprises in installation, configuration, or use.

The Networks of Workstation (NOW) [1] project at UC Berkeley was focused more on system-level research rather than supporting scientific computing applications. Sandia National Labs has the Computational Plant project, which is constructing multicomputers from DEC Alpha nodes. However, they have ported their custom OS (Puma) to the Alpha, with the intent of providing ASCI Red-like system functionality on a COTS base.

In our case, we focused on scientific computing and high performance using commodity components and OS software. Thus, we have made different choices than each of the projects mentioned above, although Centurion shares some common ground with each.

5   System Component Performance

Before we examine application performance on our cluster, we will describe microbenchmark performance measurements taken on components of the cluster, including the interconnection networks and the various processors. These microbenchmarks are important so that we can properly set our expectations for application performance.

5.1   Myrinet performance

Myrinet offers 2 different interfaces to the system; the usual "networking" interface via the kernel running protocols such as TCP and UDP, and a user-level interface similar to U-Net [2] or the VIA standard [4]. Gigabit networking requires a low-overhead interface in order to get gigabit performance; there have been many demonstrations of 700 to 900 gigabit sustained transfers using Myrinet with various user-level interfaces. However, we have yet to write an interface between the Legion networking code and a user-level Myrinet interface, so we are still using the kernel interface. This interface gives us a raw performance (measured by netperf) of 240 megabit/s for both TCP and UDP. This compares favorably with the loopback result of 650 megabit/s and 850 megabit/s for TCP and UDP. TCP latency was 390 usec, compared to 60 usec for the loopback.

As mentioned earlier our Myrinet setup has a theoretical bisection bandwidth of 32 gigabits per second, and we were unable to produce a bottleneck in the network under any

load we could generate.

5.2 Ethernet performance

The single-node netperf results are roughly 70 megabits for TCP and UDP. Ideally we should be able to hit near 100 megabits, but the drivers are often unable to do that, and our network is not quiet. TCP latency was 103 usec to a node on the local hub, and 130 usec to a node on a remote hub. Note that the latency numbers are lower than Myrinet; we attribute this to high overhead in the Myrinet kernel driver, because we know that user-level Myrinet protocols can achieve sub-10 usec latencies [6].

The performance of our cluster using the cheap semi-switched 100 megabit Ethernet topology described earlier is hard to quantify. While an Ethernet segment with one sender and one receiver can achieve full wire speed, our experience shows that segments with many senders and receivers may only achieve 30% of the theoretical maximum, and latencies become substantial. This is, essentially, similar performance to traditional Ethernet in terms of relative saturation levels; given that the lowest branch of our cluster topology are based on unswitched Ethernet, this is not at all surprising. The measured bisection bandwidth (each system sending to a system on a different hub) is 400 megabit/s. The bisection bandwidth of today's large 100 megabit switches ($400/node as opposed to $110/node) is often bound by limited backplane bandwidth to a few gigabits; as you can see, this is far better than our semi-switched (but cheap) network.

5.3 Stone soup performance

In addition to the 64 alphas we have been discussing thus far, we already owned 16 dual-processor 200mhz Pentium Pro machines, all of which have Myrinet cards. These machines are mostly used by our developers, but we do heterogeneous computations using Legion on these machines and Centurion on a regular basis.

The issue of compilers is always a vexing one for Linux systems. For the x86, the compilers from the Portland Group have published SPEC numbers that are slightly higher (under Linux) than Intel's best fp results; our running of gcc-2.7.2 is 22% slower than Intel's best result in integer, and 23% slower in fp[2]. Intel has made optimized BLAS routines for Linux available via public ftp as part of the ASCI Red project. We have a long-standing relationship with Sandia's MPCRL (the people who do the OS support for ASCI Red), so we expect to use these optimized routines in the future to maintain compatibility with work at the MPCRL.

5.4 Alpha performance

The issue of compilers is doubly interesting on the alpha, because it was known that gcc does a poor job of generating code for the alpha. However, our running of SPEC95fp with gcc/egcs-1.0.1 showed that gcc was between 7% faster and 30% slower than Digital's

---

[2]full details available upon request

compiler. (This SPEC result is not formally publishable because SPEC has never bothered to approve how we compiled 'make', but we are happy to provide the details upon request.) It is also possible to cross-compile with Digital's compiler on a Digital Unix machine and run the resulting (statically-linked) binaries under Linux.

In addition to compiler quality, library quality is also an issue. As usual this is a work in progress; good fundamental math library routines for the Alpha will be released in July 1998, and an optimized BLAS matrix-multiply routine written by a patent clerk in Japan (K. Goto) is faster than Digital's BLAS routines, but not all BLAS routines are available in optimized form under Linux.

6 Applications

To illustrate the use of Centurion to support scientific applications, we will describe our experiences in porting four separate applications to the machine. We have selected these programs as a representative sample of the application universe in which we work; the suite represents a computational fluid dynamics code, generic parameter-space studies in Fortran, a macro dataflow program for genomics, and a particle-in-cell code used in materials science.

6.1 Nameless Ocean Model (NOM)

As part of our participation in the Department of Defense High-Performance Computing Modernization (HPCMOD) program, we have helped the NAVO MSRC[3] conduct benchmarks on various hardware. The "Nameless Ocean Model" is one program used for this purpose. This program solves shallow-water equations; this sort of code is used today to simulate shallow estuaries and in modeling of oceans and ice in the polar regions. It is a Fortran program which uses a time-explicit finite difference technique, which can be thought of as applying an expensive stencil operation repeatedly to all points on a grid. Computational fluid dynamics (CFD) programs are often thought to require such low latencies and high bandwidth communications that they are not suitable for clusters of machines, but we will show that this particular code runs well on our cluster.

The version of the code handed to us had already been parallelized for a homogeneous machine, using MPI message-passing. It was parallelized in the traditional way, namely the computational domain was decomposed in two dimensions into squares, and boundary regions were exchanged between adjacent squares every timestep, as shown in figure 1. Other CFD techniques such as implicit or spectral methods require more communications.

It turns out that this code runs quite well on our existing, homogeneous cluster. Using our semi-switched 100 mbit networking, the code got 155 MFlops "out of the box" on 4 cpus. After suitable cache-friendly optimizations, it got 476 Mflops on 4 cpus and 3700 Mflops on 49 cpus. This code gets around 650 Mflops on a single Cray T90 cpu. More interesting to us is a comparison to an SGI Origin 2000; we get about 70% of the O2k's performance at 49 cpus, but our machine costs 1/10 as much.

---

[3]Major Shared Resource Center–a DoD supercomputer center.
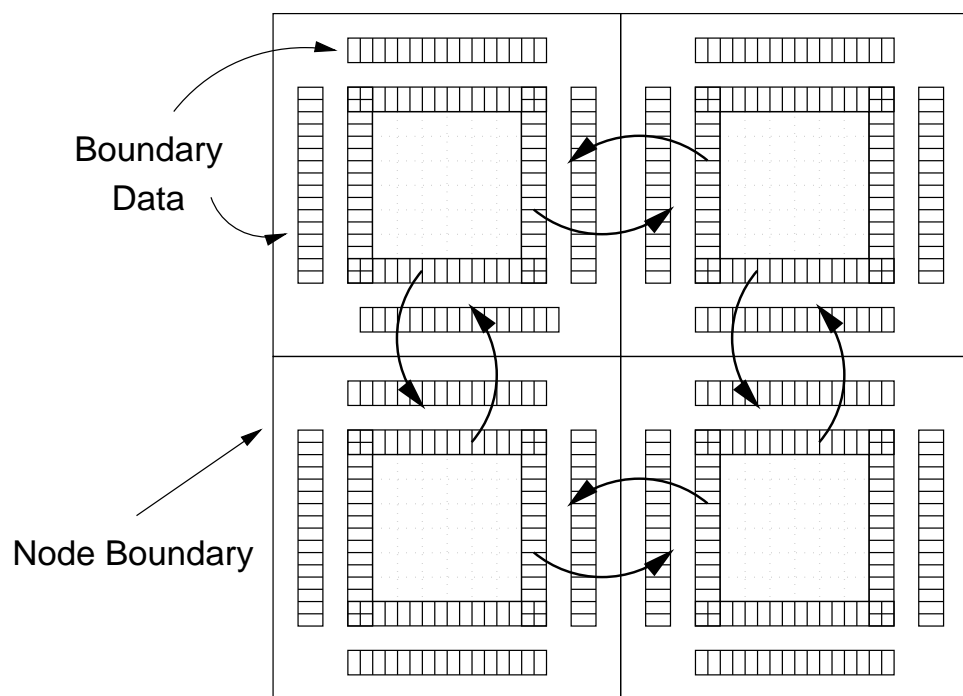
Boundary Data

Node Boundary

Figure 1: Nameless Ocean Model decomposition

In the near future, we plan on building a generic tool which can work with the application to statically allocate different-sized domains to processors of different speeds. In this way, we will take advantage of Centurion Phase II, as well as our other existing infrastructure such as the Stone Soups.

6.2 Flogger

As part of our exploration of ways to run generic, non-Legion-aware programs under Legion, we have built an example tool which allows remote, transparent execution of serial programs whose only interaction with the user is via disk files. One real-life class of applications which benefit from this is scientific parameter-space searches. For example, we might want to run several programs in sequence which compute the lift generated by a particular airplane wing at various atmospheric pressures and angles of attack, and we have thousands of combinations of pressures and angles and wing designs. Each run might take only a few hours on a single cpu, but the total cpu time consumed could be many cpu-years. This sort of problem is something we call a "bag of tasks.[4]"

Since our interests extend beyond cluster computing to metacomputing, the example tool we built used existing Legion mechanisms to securely transport both the executable program and data files to remote nodes and return the output files to the user. This is overkill on a cluster, but is very useful when the resources used by a computation span several administrative domains, which do not share NFS-mounted filesystems.

To date, we have used this example tool to run "bag of tasks" computations which transparently used Centurion and machines (with a different architecture) at the San Diego Supercomputer Center. The example tool consists of 163 lines of perl code.

6.3 Complib

An on-going research project which pre-dates the Legion effort is the Mentat Programming Language, a C++ superset which supports coarse-grained parallel processing. Mentat allows the programmer to declare instances of particular C++ classes to represent objects, which execute in separate addresses spaces. The Mentat compiler and run-time system then performs macro-dataflow on the inputs and results of method calls on these objects, automatically building graphs of inter-related method calls and executing each method call as its inputs become available.

We have used this language to construct an application for DNA and protein sequence comparison, called Complib. Biologists determine the structure of new proteins by comparing their sequences to a library of known proteins. This algorithm maps well to the notion of macro-dataflow. The MPL code for the fundamental kernel of the program is shown in figure 2. The match function computes an integer result which represents the degree of similarity of the two genetic sequences (the test sequence and one from the library). This function is stateless; thus, we know that all iterations of this loop could be performed in

---

[4]The name implies that one just reaches into the bag and grabs the next task to be run; tasks are completely independent, and order of execution is not important.

```
mentat stateless compute;
mentat collector;

do i = 1, n
  do j = 1, m
     collector.result(compute.match(sequence(i),sequence(j)))
  enddo
enddo
```
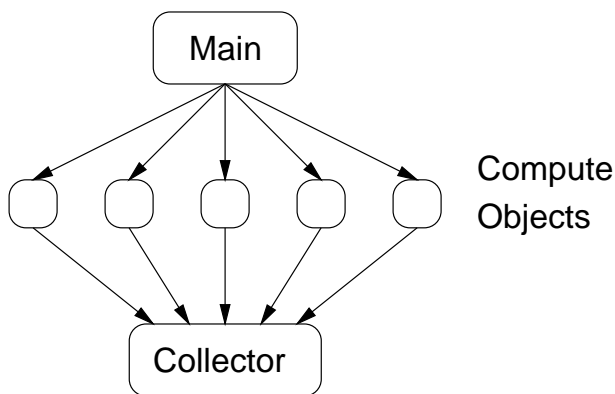
Figure 2: MPL program for Complib

Figure 3: The Macro-Dataflow graph for Complib

parallel. The Mentat compiler is able to recognize this property in the Mentat program shown in figure 2. From this code, the Mentat run-time system generates the graph shown in figure 3.

The underlying Legion run-time system has the opportunity to decide the number of "compute" objects created to process this loop, and to use the Legion resource management facilities for object placement. Once the worker objects are placed, we can load balance the computation by directing method calls to the worker objects with the lightest load (smallest number of outstanding requests).

6.4   Particle-in-cell Code

Another problem run on Centurion was a particle-in-cell code which is used to simulate particle deposition onto a surface. In this simulation, the surface of interest is broken down into small areas called cells. Each cell tracks the deposition of a number of particles (hence the name). One application of this code is modeling the production of VLSI chips

by ion deposition. We were presented with a serial version of this code, and parallelized it in the obvious fashion, decomposing the spatial domain, having neighboring domains exchange messages. Even though the amount of computation per particle was fairly small, the number of particles needing to be exchanged was small enough that the code runs quite well on our 100 mbit semi-switched Ethernet network.

As seen in figure 4, we observed super-linear speedup. We attribute this to a lack of sufficient cache when only a few processors were used. The data represent a variety of spatial decompositions; if the number of rows is > 1, this indicates a 2-dimensional decomposition. Full details are available in [3].

7   Concluding Remarks

In this paper, we have described some of our experiences building the Centurion cluster, and porting and tuning applications to our run-time system using the cluster. Our experiences in building the cluster demonstrate that a high-performance multicomputer can be constructed from inexpensive commodity parts. Our experience moving a range of applications to Centurion leads us to conclude that such a platform is well-suited to many applications, even some which have historically been regarded as impractical for distributed-memory multicomputers.

In short, we have demonstrated that with proper care and feeding, a COTS cluster can outperform dedicated supercomputers at a fraction of the cost. A system capable of sustaining computation rates comparable to commercial supercomputers can be had for a fraction of the cost, and Moore's law dictates that the price-performance ratio on these machines will continue to grow faster than that for dedicated supercomputers.

References

[1] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW Team. A case for networks of workstations: Now. *IEEE Micro*, February 1995.

[2] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, December 3–6 1995.

[3] N. Beekwilder and A. Grimshaw. Parallelization of an axially symmetric steady flow program. Technical Report CS-98-10, Department of Computer Science, University of Virginia, May 1998.

[4] Intel Corporation. The virtual interface architecture. http://www.intel.com/procs/SERVERS/isv/vi/vi2/index.htm.

[5] A. S. Grimshaw, Wm. A. Wulf, and the Legion Team. The legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1), January 1997.

Figure 4: Measured speedup for particle-in-cell code

[6] S. Pakin, V. Karamcheti, and A. Chien. Fast messages: Efficient, portable communication for workstation clusters and MPPs. *IEEE Concurrency*, 5(2), April-June 1997.

[7] T. Sterling. Low cost (m2cots) cluster system design and development (derived from the beowulf experience). http://www.cacr.caltech.edu/ tron/presenta.htm.