Application Performance of a Linux Cluster using Converse [*]

Laxmikant Kalé, Robert Brunner, James Phillips and Krishnan Varadarajan
kale@cs.uiuc.edu, {brunner,krishnan,jim}@ks.uiuc.edu

Department of Computer Science, and
Beckman Institute of Advanced Science and Technology
University of Illinois at Urbana-Champaign

Abstract. Clusters of PCs are an attractive platform for parallel applications because of their cost effectiveness. We have implemented an interoperable runtime system called Converse on a cluster of Linux PCs connected by an inexpensive switched Fast Ethernet. This paper presents our implementation and its performance evaluation. We consider the question of the performance impact of using inexpensive communication hardware on real applications, using a large production-quality molecular dynamics program, NAMD, that runs on this cluster.

1 Introduction

PCs, as commodity components, have always been fairly inexpensive. Over the past few years, PCs have gained significantly in performance even as their price continues to drop. As a result PCs make an attractive platform for computational programs. By connecting many PCs together in a cluster, one can build a "parallel machine" for a fraction of the cost of dedicated parallel machines such as the Cray T3E, SGI Origin 2000, or the ASCI Red (a one of a kind machine built by Intel, for the U.S. Department of Energy). The Beowulf project [12] exemplifies this cost effective approach to parallel computing. Several earlier parallel programming systems, such as Linda [1], Charm [2] and PVM [13] also supported clusters of workstations.

Such commodity clusters can either use commodity communication hardware, such as switched Fast Ethernet, or hardware designed especially for high-performance computing, such as Myrinet or switches based on the VI Architecture standard. Significant research has been carried out [10] in improving communication speeds using such dedicated hardware. Although such hardware improves communication speeds dramatically, at this point in time it is considerably more expensive. The large number of PCs on peoples' desks makes PC

[*] This work was supported by the National Institutes of Health (NIH PHS 5 P41 RR05969-04) and the National Science Foundation (NSF/GCAG BIR 93-18159 and NSF BIR 94-23827 EQ). James Phillips was supported by a Computational Science Graduate Fellowship from the United States Department of Energy.

clustering an attractive platform for parallel applications. As most of the compute power available on the desktop typically remains unused (especially after work hours, but even during the workday, since most user applications do not require much computational power) parallel applications can be deployed on such platforms without significant up-front costs. Such environments typically have only commodity communication hardware.

The questions that arise in this context are: Is a PC cluster with commodity communication hardware a viable parallel platform? What is the extent of communication performance loss with such hardware? And, most importantly, what is the impact on real applications of this low communication performance? We attempt to answer these questions in this paper.

As a part of a large collaborative research project, we are developing a parallel molecular dynamics program, called NAMD [6, 9]. Although the program runs on dedicated parallel machines at the national centers, we needed a platform we could use continuously. We have built a cluster of 32 Intel processors (16 2-way SMP nodes), connected by a 100 Mbps switched Fast Ethernet. We ported the Converse [5] run-time system to this platform. In this paper, we first describe the Converse port, and its raw communication performance. We then examine the impact of communication performance on a production application, NAMD.

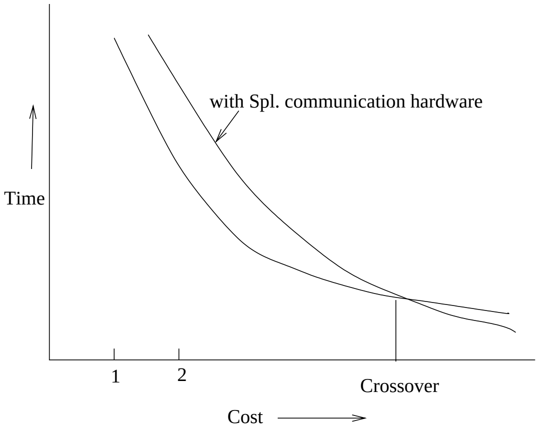

Fig. 1. Cost and Performance of systems with and without specialized communication hardware

The broader issue that this study raises is whether special purpose communication hardware is necessary for cost-effective performance on various applications. In general, the answer to this question depends on the number of processors to be used, and the characteristics of the application. Figure 1 shows a schematic plot of time-to-completion for an application versus the cost of the hardware, for (a) a simple system using commodity hardware (such as switched

Ethernet) and (b) a more expensive one using specialized communication hardware (such as Myrinet switches). On one processor, clearly both systems perform equally, but (b) costs more. On a very large number of processors, for most applications, (b) will perform better when comparing identical cost configurations. Somewhere in the middle, the curves cross over. The last section of our paper studies the factors that affect this crossover point.

2 Converse on the Linux cluster

Converse is a run-time framework that we have developed to simplify implementations of new parallel languages, and to support interoperability among them. Converse has a component based architecture, including modules for interprocess communication, user-level threads, prioritized scheduling, and load balancing. Using a common scheduler for entities across different parallel paradigms/languages, such as threads, handlers, and data-driven objects, Converse allows multiple language modules to interoperate concurrently. Its component architecture allows one to put together run-time systems for new languages quickly, without losing performance. Converse supports portability, and is available on major parallel machines and clusters of Unix workstations.

For the cluster of PCs, we adapted the workstations version of Converse. As we did not want to limit the number of PCs connected in a cluster, we use the connection-less UDP protocol. The user's message is divided into fixed-size packets. As UDP packet delivery is not guaranteed, we implement a window-based flow-control protocol. To deal with the fact that there are two SMP processors on each PC, we use a single socket per PC. A producer-consumer queue is used for each processor to store its incoming messages. Since there is exactly one producer and one consumer, this queue is implemented without using any locks. Messages between two processors within the PC are transmitted using the shared memory, avoiding the socket.

2.1 Communication Performance

To analyze the raw communication performance of our implementation, we used a simple ping-pong program. The results are shown in Figure 2. A short message takes about 250μsec. for its application-to-application delivery across the network. The incremental cost for each additional byte is approximately 130 nanoseconds, leading to the bandwidth of about 7.4 MB/sec (59 Mbits/sec). The communication bandwidth attained in this benchmark is also shown in the same Figure, as a function of message size. Figure 2 indicates that we attain half the peak bandwidth for 2 KB messages. Similar data for large messages is shown in Figure 3. We believe that our implementation can be further optimized to some extent, but we expect the attained bandwidth to be limited to about 10 MB/sec.

To compare this performance with a dedicated parallel machine, we present Converse communication performance on the ASCI Red machine in Figure 4. Latency of around 45μsec, and bandwidth of 50 MB/sec is attained.

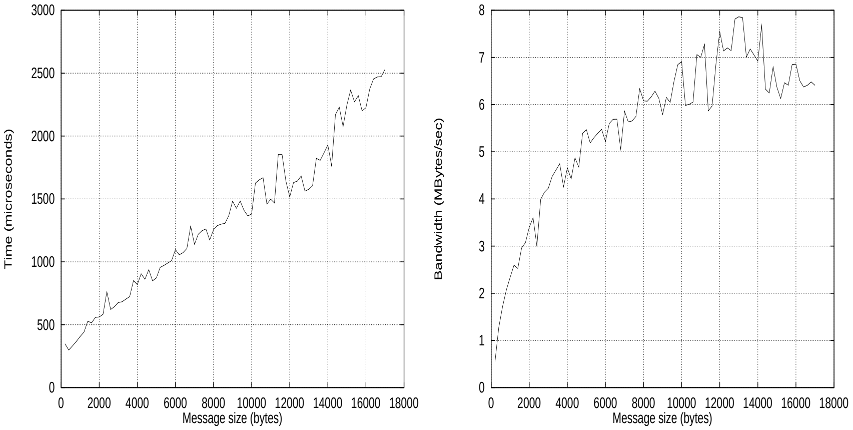

Fig. 2. Raw message time (one-way) and bandwidth for the ping-pong benchmark on the Linux cluster.

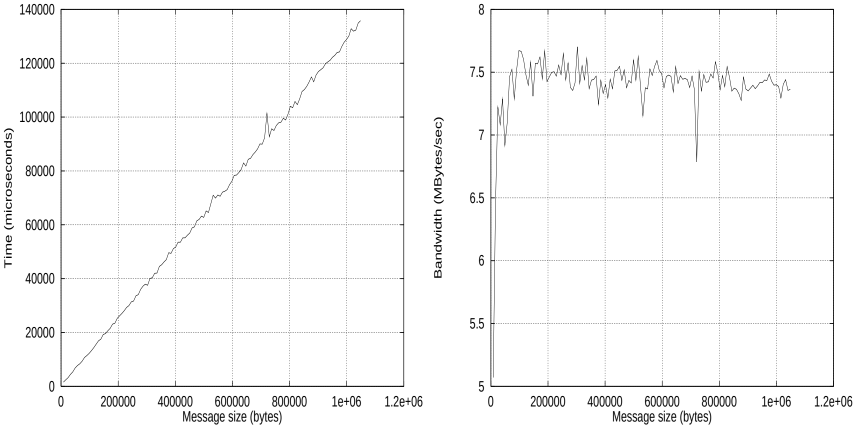

Fig. 3. Raw message time (one-way) and bandwidth for large messages on the Linux cluster.

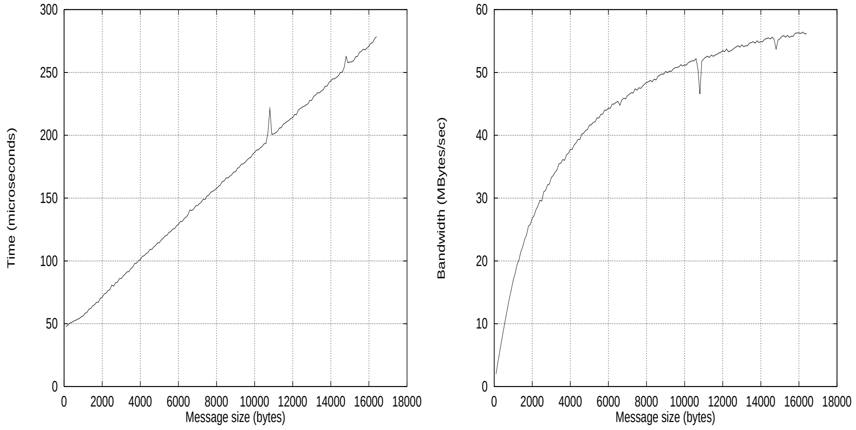

Fig. 4. Raw message time (one-way) and bandwidth for the ping-pong benchmark on the ASCI-Red.

On PC clusters, Myrinet based communication systems (such as FM [8], which was one of the earliest) provide a much higher performance compared with our switched Ethernet implementation. One of the fastest, for example, is BIP [10], which supports a less than 5μsec. latency, and 120 MB/sec bandwidth. So, while it is still important to try improving the performance on the commodity hardware, it is worthwhile asking whether this big a disparity in communication performance renders parallel applications infeasible. To answer this question, we turn to a study of a full-fledged application.

3 Parallel Molecular Dynamics: NAMD

Molecular dynamics programs simulate the motions of molecules by repeatedly computing the forces on all the atoms and numerically integrating the equations of motion. Our users are interested in using molecular dynamics simulation to study the behavior of biochemically significant molecules such as proteins, membranes and DNA. Typical simulations are composed of one thousand to one hundred thousand atoms, and require simulation times of hundreds of picoseconds. Since the numerical characteristics of such simulations mandate timesteps of the order of one femtosecond, a complete simulation requires several hundred thousand timesteps. Each timestep requires several CPU-seconds of computation, so sequential simulations would require weeks or months to complete. Parallel simulation is almost mandatory for such simulations, but parallelizing the integration algorithm is complicated since each timestep must be completed on every processor before the next timestep can begin. Therefore, efficient parallel simulation requires parallelizing operations within a single timestep which only requires a few seconds to compute sequentially.

NAMD is a parallel molecular dynamics program, designed to be scalable [6]. It is implemented using Charm++ [7] and Converse. It uses aggressive parallelization strategies, and an object-based load balancing strategy that relies on run-time measurements for accurate remapping of objects. NAMD is a relatively irregular program, with thousands of concurrent objects, each representing widely different computational load, and with complex communication patterns. It is one of the fastest production-quality parallel molecular dynamics program, and several scientific simulation studies of biomolecules have been conducted using it. NAMD relies on the message-driven communication model provided by Converse to adaptively overlap computation with communication. It also uses the multilingual capabilities of Converse to incorporate the DPMTA PVM-based full-electrostatics library from Duke University [11].

The current version of NAMD is optimized to run efficiently on dedicated parallel machines. This required optimizing the communication so that each timestep can be completed in less than a second on machines with hundreds of processors. The parallel molecular dynamics algorithm has the following outline:

1. Distribute atom positions from owning processors to several other processors which must compute atom forces.
2. Compute forces based on incoming atom positions.
3. Return forces to processors which own those atoms.
4. Update atom positions based on incoming forces.
5. Return to the first step.

NAMD uses a hybrid spatial/force decomposition scheme. Atoms are not owned by processors, but by objects called *patches*, which are distributed among the processors. Each patch is responsible for the atoms in a cubical region of the simulation space. Each patch sends requests to a set of *compute* objects, which compute the forces exerted on the atoms of that patch by other atoms in the simulation. Unlike in a pure spatial decomposition scheme, a compute object may reside on a different processor from its associated patches, in which case the atom positions from a remote patch are stored in a *proxy patch.* This minimizes communication for the case where several compute objects on a processor are accessing the patch data; the atom positions are sent to the processor only once per timestep, and the returned forces from each compute are combined and sent as a single message.

Allowing compute objects to reside on any processor, independently of where its patches reside, provides two important capabilities. First, NAMD can utilize parallel machines with more processors than patches. Second, compute objects can be shifted around to achieve good load balance. NAMD uses a run-time, measurement-based load balancing algorithm, that measures how much computation time each compute object consumes, and then periodically rebalances the compute objects to minimize idle time while simultaneously keeping communication overhead small.

NAMD is a production-quality application currently being used for several simulations. It is freely distributed. More details are available at the NAMD web site: `http://www.ks.uiuc.edu/Research/namd/namd.html`.

3.1 NAMD performance

NAMD has achieved its design goal of scalable performance on large parallel machines. For example, it yielded a speedup of over 180 using 220 processors. Efficient scalability to large numbers of processors required us to carefully optimize communication patterns in the program. These communication optimizations have had the unintended side effect of producing good speedups on smaller machines with relatively poor communication performance, such as the Linux/Fast Ethernet cluster.

Once Converse was ported to the cluster, NAMD was ported relatively effortlessly. Table 1 shows the performance of NAMD on the PC cluster (400MHz Pentium II), and compares it with its performance on two dedicated parallel machine, the Cray T3E (450MHz Alpha) and the Intel ASCI Red (200MHz Pentium Pro). Despite the irregular and dynamic nature of the parallel computation, the adaptive load balancing strategies and prioritized scheduling techniques in Converse lead to speed up of about 14.7 on 16 processors of this dedicated machine. The speedup using the cluster is about 12.2 on 16 processors, and 19.9 on 32 processors. Thus, in spite of the huge communication penalty, one is able to derive useful speedups on this cluster.

There are two separate conclusions one can derive from this experience. Firstly, it is clear that an existing network, not designed for parallel processing, can be used profitably for parallel applications. The second question is more subtle: is the investment in extra communication hardware worthwhile? Given NAMD performance on dedicated parallel machines, we expect the speedup would be around 28 on 32 PEs using specialized communication hardware. This represents about forty percent increase in performance over the Linux cluster performance on 32 processors. The current cost of additional communication hardware is between 25 to 40 percent of the cost of the PCs themselves. So, additional hardware may be a worthwhile investment. However, on 16 or fewer processors, additional hardware is clearly not cost effective. Even on 32 processors, planned optimizations will narrow the gap further. Another implication of this result is that although it is intellectually challenging to improve the raw communication cost, the effort will not lead to a commensurate performance improvement in real applications.

Furthermore, the execution times point out another advantage the clusters have over dedicated machines. The latest processor technology can be quickly integrated into a cluster, whereas a dedicated parallel machine is somewhat slower in incorporating newer processors. Although the ASCI Red used the fastest Intel processors available when it was designed, they are now less than half the speed of commonly available Linux machines.

4 A Controlled Benchmark

For our molecular dynamics program, we observed that beyond twenty processors, extra communication hardware becomes cost effective. However, we cannot

		Processors								
		1	2	4	8	16	20	24	30	32
T3E	Time		6.12	3.10	1.60	0.810				0.397
	Speedup		(1.97)	3.89	7.54	14.9				30.3
ASCI Red	Time	28.0	13.9	7.24	3.76	1.91				1.01
	Speedup	1.0	2.01	3.87	7.45	14.7				27.9
LINUX	Time	12.34	6.42	3.28	1.69	1.01	0.85	0.74	0.65	0.62
	Speedup	1.0	1.92	3.77	7.30	12.21	14.51	16.68	18.98	19.90

Table 1. Execution time (sec./timestep) for ER-ERE (36,573 atoms, 12Å cutoff).

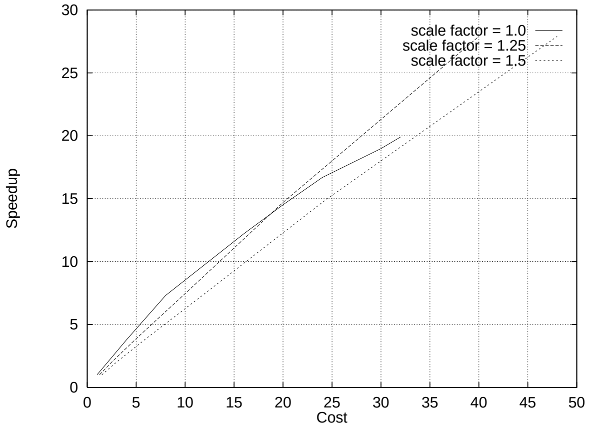

Fig. 5. NAMD Speedup, adjusted for communication cost.

generalize this observation to all application. The crossover point will depend on the characteristics of the particular application. More specifically, it depends on the communication load and how it scales with the number of processors in a given application.

To examine the effect of communication performance on the overall performance, and its relationship to the cost of the entire system, we studied a simple, well-controlled benchmark. The program used for the study is Jacobi relaxation using a five point stencil. The data for this program is a square array, which is partitioned across all the processors. Thus, each processor itself has a square piece of the array, assuming the total number of processors is a square. In each iteration, a processor sends the boundary elements (either a row or a column) to each neighboring processor. Thus, except for the processors on the boundary, each processor sends and receives four messages in each iteration. As the amount of computation per processor is proportional to the size of its partition, while the communication is proportional to the size of its boundary, we can control the computation to communication ratio by varying the size of the data array.

To compare the performance of this program on an architecture with fast communication hardware, we used the following procedure. First, we ran the program on the Cray T3E. To account for differences in absolute processor speed between the T3E and the Linux cluster, we compare speedups rather than absolute speeds. Thus, we are assuming that by using a more expensive communication hardware with Linux, we would be able to achieve speedups similar to those on a tightly coupled machine (T3E). To compare cost performance, we now plot the speedups as a function of cost. For concreteness, we assume that specialized communication hardware will cost fifty percent more than the cost of each processor with only commodity communications. The speedups for each problem size are plotted in Figure 6. Rather than plotting speedup against number of processors, the x-axis of these plots is *cost*, where *cost* = *# of processors* $*$ *scale factor*. For example, the cost of two processor with commodity communications is 2.0, and the cost of two processor with specialized communications is 3.0.

For an application group deciding between building PC cluster with or without a high-cost communication network, can we draw any lessons from these experiments? Consider how the performance of Jacobi behaves in Figure 6 as the number of processors increases. A simple analysis shows that for 16 processors, each processor sends an average of 3 messages per iteration (48 messages total). For 25 processors, the above analysis gives almost the same number of messages, an average of 3.2 messages per iteration (80 messages total). For the grids examined, each message is only a few hundred bytes long. We know from the communication studies in earlier sections that for these messages, the α component (per message cost) dominates over the β component (per byte cost), so the communication time is determined by the number of messages. For a fixed grid size the computation scales perfectly with the number of processors. Therefore the computation to communication ratio decreases, making expen-

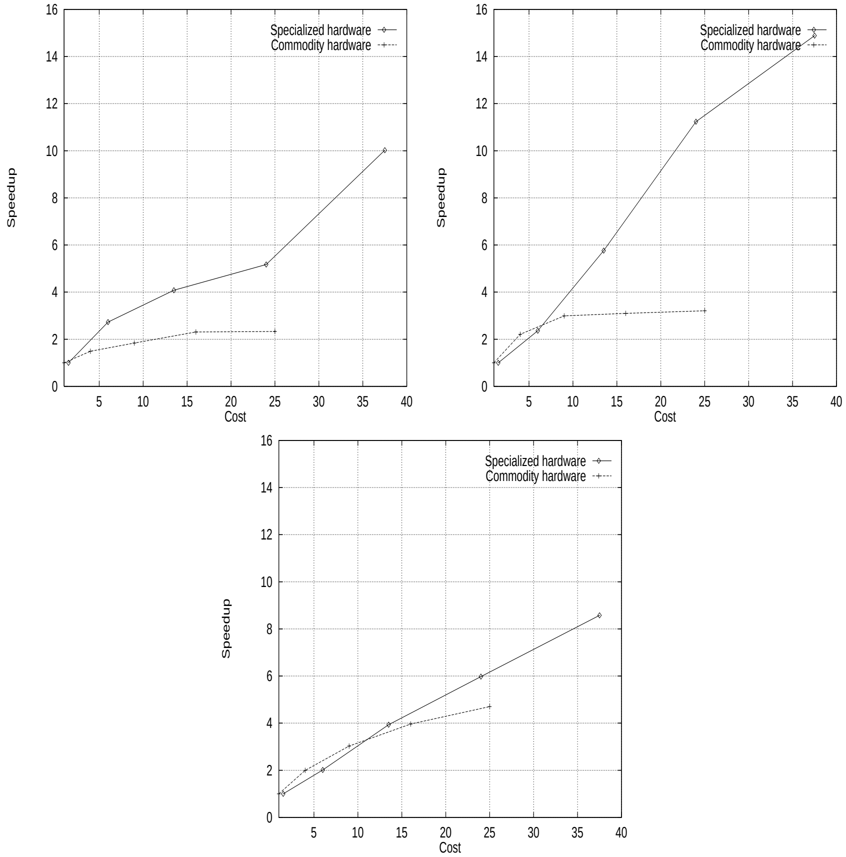

Fig. 6. Speedup versus cost for the Jacobi benchmark (Top left : 240×240 grid, Top right: 360×360 grid, Bottom: 480×480 grid)

sive communication hardware a more desirable alternative to larger numbers of processors.

Consider also the three problem sizes shown in Figure 6. Each problem size represents a different "application" with different computation to communication ratios. Clearly, the crossover point depends on the computation to communication ratio. As the grid size increases, the computation time increases. As described above, the communication time for these grid sizes is determined by the number of messages. Since the number of messages is fixed as the grid size changes, the communication time remains constant. Therefore the speed of communication hardware becomes a less significant factor in program performance, and the crossover point shifts to the right.

In general, one must study how the number of messages, m, and the number of bytes communicated per processor, n, varies as a function of the total number of processors used, p. Let us assume these characteristics are given by functions $m(p)$ and $n(p)$ respectively. Also, let α_1 and α_2 be the per message costs on the commodity and special-purpose hardware respectively. Similarly, let β_1 and β_2 be the per byte cost of communication.

For applications like Jacobi, the computation scales perfectly. So, if T_c is the computation time on 1 processors, the computation time on p processors will be T_c/p.

Let r be the fractional cost of communication hardware per processor, so that the communication hardware cost per processor $= r *$ (cost per node). Given a fixed budget, one has a choice of spending it on p_1 processors or $p_2 = p_1/(1+r)$ processors with expensive communication hardware.

Now that we are comparing equal cost configurations, the total time to completion, T_{total} for each can be calculated, as

$$T_{total} = \alpha_1 * m(p_1) + \beta_1 * n(p_1) + \frac{T_c}{p_1} \tag{1}$$

and

$$\begin{aligned} T_{total} &= \alpha_2 * m(p_2) + \beta_2 * n(p_2) + \frac{T_c}{p_2} && (2) \\ &= \alpha_2 * m(p_2) + \beta_2 * n(p_2) + \frac{T_c(1+r)}{p_1} && (3) \end{aligned}$$

Equations 1 and 3 clearly show the tradeoff involved: although the communication terms (α_2 and β_2 terms) are smaller for the expensive architecture, the computation time may increase because we can afford only fewer processors. The important point is that the switch-over critically depends on the nature of the functions $m(p)$ and $n(p)$, which can only be characterized through careful analysis of the particular application.

For application where the computation time itself does not scale well, due to load imbalances, critical path effects, or algorithmic overheads, a more refined analysis must be conducted to determine the crossover point. In such situations the crossover point will be pushed in favor of communication hardware, due to the higher efficiency attained on fewer processors.

5 Conclusion

A PC cluster is indeed a cost-effective platform for parallel applications. However, the question of whether to use special-purpose communication hardware (such as Myrinet) turns out to be somewhat complex. Our experience with NAMD, a production-quality molecular dynamics program, demonstrated that commodity hardware (such as switched 100Mb/sec Fast Ethernet) is adequate to provide good speedups up to 16 processors. Although the speedup on 32 processors was around 20 with the cluster, as compared with 28 on a dedicated parallel machine, one must take the cost of additional communication hardware into account. We showed a simple method for cost-performance analysis that can be used to decide whether to go for high-cost communication hardware. The crossover point beyond which it becomes profitable to deploy such hardware was seen to depend on specific communication-related characteristics of the application. Once the communication scaling behavior of the application is understood, one can simply plug in the appropriate values in expressions given in the paper, to determine which configuration will be appropriate for a given budget.

Success attained by the Avalon project [14], with a cluster of Alpha workstations connected by relatively inexpensive Fast Ethernet, indicates that for several applications one may be able to benefit by using commodity communication hardware. Of course, with the emergence of VI Architecture standard, and results such as those presented in this paper, coupled with growing popularity of clusters, the cost of special purpose hardware itself may drop substantially. However, this will be offset by further reductions in microprocessor costs. A careful cost analysis may still be needed in the future.

The Converse system itself is neutral with respect to communication technology. Converse runs on almost all available communication hardware, including ATM, shared Ethernet, switched Ethernet, and Myrinet. So, programs written using Converse will be able to exploit any emerging technology.

6 Acknowledgements

The computer time for the performance comparisons on the ASCI Red was provided by the Center for Simulation of Advanced Rockets at the University of Illinois at Urbana-Champaign, and the Accelerated Strategic Computing Initiative of the Department of Energy. The T3E time was provided by the Pittsburgh Supercomputing Center.

References

1. N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32(4), April 1989.
2. W. Fenton, B. Ramkumar, V. Saletore, A. Sinha, and L.V. Kalé. Supporting machine independent programming on diverse parallel architectures. In ICPP91 [3], pages 193–201.

3. *Proceedings of the 1991 International Conference on Parallel Processing*, St. Charles, IL, August 1991.
4. L. V. Kalé, Milind Bhandarkar, Robert Brunner, N. Krawetz, J. Phillips, and A. Shinozaki. NAMD: A case study in multilingual parallel programming. In *Proc. 10th International Workshop on Languages and Compilers for Parallel Computing*, pages 367–381, Minneapolis, Minnesota, August 1997.
5. L. V. Kalé, Milind Bhandarkar, Narain Jagathesan, Sanjeev Krishnan, and Joshua Yelon. Converse: An Interoperable Framework for Parallel Programming. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 212–217, Honolulu, Hawaii, April 1996.
6. Laxmikant Kalé, Robert Skeel, Milind Bhandarkar, Robert Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *J. Comp. Phys.*, 1998. In press.
7. L.V. Kale and S. Krishnan. Charm++: A portable concurrent object oriented system based on C++. In *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, September 1993.
8. S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of Supercomputing 1995*, dec 1995.
9. James C. Phillips, Robert Brunner, Aritomo Shinozaki, Milind Bhandarkar, Neal Krawetz, Laxmikant Kalé, Robert D. Skeel, and Klaus Schulten. Avoiding algorithmic obfuscation in a message-driven parallel MD code. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. Mark, S. Reich, and R. D. Skeel, editors, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, volume 4 of *Lecture Notes in Computational Science and Engineering*, pages 455–468. Springer-Verlag, November 1998.
10. L. Prylli and B. Tourancheau. BIP: A New Protocol Designed for High Performance Networking on Myrinet. *Lecture Notes in Computer Science*, 1388:472–??, 1998.
11. W. Rankin and J. Board. A portable distributed implementation of the parallel multipole tree algorithm. *IEEE Symposium on High Performance Distributed Computing*, 1995. [Duke University Technical Report 95-002].
12. T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF : A parallel workstation for scientific computation. In *International Conference on Parallel Processing, Vol.1: Architecture*, pages 11–14, Boca Raton, USA, August 1995. CRC Press.
13. V.S. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, 2(4), December 1990.
14. Michael S. Warren, Timothy C. Germann, Peter S. Lomdahl, David M. Beazley, and John K. Salmon. Avalon: An alpha/linux cluster achieves 10 gflops for $150k. Technical report, Los Alamos National Laboratories, http://loki-www.lanl.gov/papers/sc98, SC98 Gordon Bell Award Entry, 1998.