Evaluation of Cluster Interconnects for a Distributed Shared Memory *

Sumit Roy and Vipin Chaudhary
Parallel and Distributed Computing Laboratory
Wayne State University
Detroit, MI – 48202
sroy@ece.eng.wayne.edu

Abstract

Clusters of Symmetrical Multiprocessors (SMPs) have recently become the norm for high-performance computing solutions, at low-costs. When using such clusters for distributed computing, the network that connects the different nodes in the cluster can become a deciding factor in the overall performance. We have developed a multithreaded Distributed Shared Memory system *Strings* and evaluated its performance on such a cluster, with different communication hardware.

The interconnection technology used include 10 Mbps Ethernet, 100 Mbps Ethernet and 155 Mbps ATM. The performance of the different configurations are evaluated using ten different programs, including kernels and applications from the SPLASH-2 benchmark suite, an application from medical computing, and some computational kernels. It is found that half of the programs tested are fairly bandwidth tolerant, and show only small performance differences when using the various networks. Though most of the applications studied are latency tolerant, the performance improvement with ATM is not very high. On the other hand, applications that do not require very high bandwidths perform substantially better with 100 Mbps Ethernet.

The paper also studies the raw performance of the networking technologies using the *netperf* evaluation tool, and compares these results with those obtained from the benchmark programs. Based on this we propose that higher bandwidth technologies alone are not sufficient to obtain good performances in such an environment.

1 Introduction

Though current microprocessors are getting faster at a very rapid rate, there are still some very large and complex problems that can only be solved by using multiple cooperating processors. These problems include the so-called *Grand Challenge Problems*, such as Fuel combustion, Ocean modeling, Image understanding,

*This research was supported in part by NSF grants MIP-9309489, EIA-9729828, US Army Contract DAEA 32-93D004 and Ford Motor Company grants 96-136R and 96-628R

and Rational drug design. Recently many vendors of traditional workstations have adopted a design strategy wherein multiple state-of-the-art microprocessors are used to build high performance shared-memory parallel workstations. These symmetrical multiprocessors (SMPs) are then connected through high speed networks or switches to form a scalable computing cluster. Examples of this class of machines include the SGI Power Challenge Array using a HIPPI interconnect, the IBM RS/6000 SP2 series with multiple PowerPC based nodes, the Convex Exemplar series of machines, the DEC AdvantageCluster 5000 with the GIGAswitch, and the Cray/SGI Origin 2000 series with the scalable Craylink network.

Using multiple nodes on such SMP clusters requires the programmer to either write explicit message passing programs, using libraries like MPI or PVM; or to rewrite the code using a new language with parallel constructs eg. HPF and Fortran 90. Message passing programs are cumbersome to write and may have to be tuned for each individual architecture to get the best possible performance. Parallel languages work well with code that has regular data access patterns. In both cases the programmer has to be intimately familiar with the application program as well as the target architecture. The shared memory model on the other hand, is easier to program since the programmer does not have to worry about the data layout and does not have to explicitly send data from one process to another. Hence, an alternate approach to using these computing clusters is to provide an illusion of logically shared memory over physically distributed memory, known as a Distributed Shared Memory (DSM) or Shared Virtual Memory (SVM). Recent research projects with DSMs have shown good performance, for example IVY [1], Mirage [2], Munin [3], TreadMarks [4], Quarks [5], CVM [6], and *Strings* [7].

This paper evaluates the performance of a multithreaded DSM, *Strings* using three different networking technologies, 10 Mbps Ethernet, 100 Mbps Ethernet and 155 Mbps ATM. The performance of the different configurations are evaluated using programs the SPLASH-2 [8] benchmark suite, an application from medical computing, and some computational kernels. It is found that many of the programs tested do not require very high bandwidths, and show only small performance differences when using the various networks. Though most of the applications studied are latency tolerant, the performance improvement with ATM compared to the 100 Mbps Ethernet is not very high. The added cost of this network technology is does not translate into corresponding gains in program execution time. On the other hand, some applications that do not require high bandwidth perform substantially better with 100 Mbps Ethernet.

The rest of the paper is organized as follows: Section two describes the distributed shared memory system used for the evaluation. The next section discusses the experimental environment on which the evaluation was done. Section four provides details about the programs used for testing. Section five presents and analyzes the results from the experiments. The final section summarizes this paper and suggests some future work.

2 *Strings* Distributed Shared Memory System

Strings [7] is a fully multithreaded DSM based on Quarks [5]. Its distinguishing feature is that it incorporates Posix1.c threads multiplexed on kernel light-weight processes for better performance. The kernel can schedule multiple threads across multiple processors on SMP nodes, using these lightweight processes.

Thus, *Strings* is designed to exploit data parallelism at the application level and task parallelism at the DSM system level. Since DSMs share data at the relative large granularity of a page, it can happen that two processes try to write to different parts of a page at the same time. This phenomenon is known as false sharing and can lead to excessive ping-ponging of the page if only a single writer is allowed. *Strings* solves this problem by allowing multiple writers, and implements data coherency using a relaxed consistency model [9].

Early generation DSM systems used interrupt driven I/O to obtain pages, locks etc. from remote nodes. This can cause considerable disruption at the remote end, and previous research tried to overcome this by aggregating messages, reducing communication by combining synchronization with data, and other such techniques [10]. *Strings* uses a dedicated communication thread, which monitors the network port, thus eliminating the overhead of interrupt driven I/O. Incoming message queues are maintained for each active thread at a node, and message arrival is announced using condition variables. This prevents wasting CPU cycles with busy waits. A reliable messaging system is implemented on top of UDP.

Event	Request (bytes)	Reply (bytes)
Page	40	8226
Lock Acquire	36	28
Lock Grant	$\geq$ 36	28
Barrier Arrival	32	28
Barrier Release	32	28

Table 1: Typical Message Sizes in *Strings*

Table 1 shows the packet sizes that occur for common events associated with the *Strings* runtime. The page size for the evaluation environment was 8192 bytes. The system maintains a distributed locking scheme, and the current list of requests is sent to the next acquirer of a lock. Hence the lock grant message can contain a variable number of bytes depending on the queue length. The actual average packet size for an application would depend on the mix of paging, locking and barrier events.

3 Experimental Environment

All the experiments were carried out on a cluster of four SUN UltraEnterprise Servers. One machine is a six processor UltraEnterprise 4000 with 1.5 Gbyte memory. The master process for *Strings* was always run on this machine. The other machines are four processor UltraEnterprise 3000s, with 0.5 Gbyte memory each. All machines use 250 MHz UltraSparcII processors, with 4 Mb external cache. Three different networks are used to interconnect the machines in the cluster, 10 Mbps Ethernet with a generic hub, 100 Mbps Ethernet with a BayStack FastEthernet Hub, and 155 Mbps ATM with a ForeRunnerLE 155 ATM switch,

Most previous work on DSMs has presented performance in a similar environment. TreadMarks has been studied on ATM networked DECstation-5000/240s [11, 4], Pentium 166s with 100 Mbps Switched

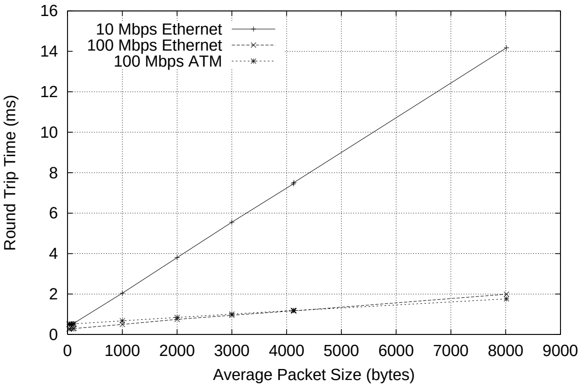

Figure 1: Roundtrip times for UDP packets

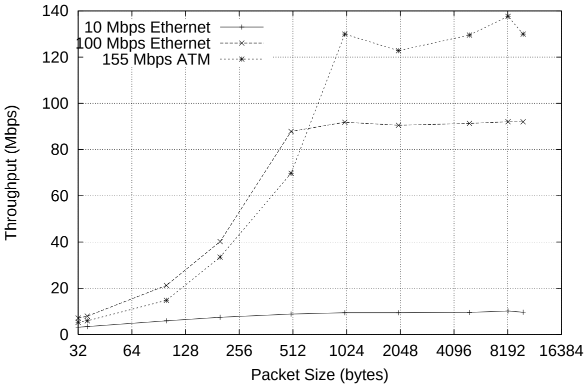

Figure 2: Throughput for UDP packets

Ethernet [12], and an 8-processor IBM SP-2 [13]. *CVM* results were presented on the IBM SP2 and a DEC Alpha cluster [14].

The raw roundtrip latencies and streaming bandwidth that can be obtained for UDP packets are shown in Figures 1 and 2. The results were obtained using two nodes on an unloaded network, using the *netperf-2.1* evaluation program [15]. The roundtrip latency includes data for typical message exchanges that would appear in the *Strings* runtime. These values place an upper bound on the performance that these networks can provide, since they do not include the effect of network collisions and congestion at the receiving ports. From Figure 1, the latencies for the 100 Mbps Ethernet and 155 Mbps ATM network indicate a crossover point at around 4000 bytes. Figure 2 shows that both 10 Mbps and 100 Mbps Ethernet can utilize the full bandwidth for packets larger than 512 bytes. The ATM network has a high startup latency, since the bandwidth utilized is less than for 100 Mbps Ethernet for packet sizes less than 512 bytes. The ATM network uses the full bandwidth for packets of approximately 1024 bytes.

4 Test Programs

The suite of test programs used to do the evaluation consists of some programs from the SPLASH-2 benchmark suite [8], a complete application program for deblurring images obtained from Magnetic Resonance Imaging (MRI), matrix multiplication (MATMULT), as well as a kernel for doing Successive Over Relaxation program (SOR) on a rectangular grid of numbers.

4.1 SPLASH-2 Benchmark Programs

The SPLASH-2 Benchmark programs have been written for evaluating the performance of shared address-space multiprocessors and include application kernels as well as full fledged code. The execution model for SPLASH-2 programs follows the Single Program Multiple Data (SPMD) type, and has three phases. In the first phase the main task reads in command-line parameters and the initial data. It sets up a shared memory region and allocates globally shared data from this region. Synchronization variables are also initialized. After this initialization phase, tasks are created to execute the actual *slave()* routine in the computation phase.

The master also runs a copy of this routine. In the termination phase the master thread collects the results and generates timing statistics. The SPLASH-2 program suite has been designed to be portable by encapsulating shared memory initialization, synchronization primitives, and task creation within PARMACS-ANL macros [16]. These macros should then be adapted for any shared memory platform.

The data access patterns of the programs in the SPLASH-2 suite have been characterized in earlier research [17, 18]. FFT performs a one-dimensional Fast Fourier Transform of n complex data points. Three all-to-all interprocessor communication phases are required for a matrix transpose. The data access pattern is hence regular. Two programs for the blocked LU factorization of a dense matrix form part of the suite. The non-contiguous (LU-n) version has a single producer and multiple consumers. It suffers from considerable fragmentation and false sharing. The contiguous version (LU-c) uses an array of blocks to improve spatial locality. The sorting kernel (RADIX) performs a radix sort on a set of integers. The program suffers from a high-degree of false sharing at page granularity during a permutation phase. Each processor writes the values of its keys into a random location in a shared array. The Ocean simulation program with contiguous partitions (OCEAN-c) simulates large scale ocean movements. This version uses a red-black Gauss-Seidel multi-grid equation solver and has a regular nearest-neighbor type access pattern. The last two programs tested evaluate the forces and potentials occurring over time in a system of water molecules. The first version (WATER-n2) uses a simple data structure, which results in a less efficient algorithm. The second version (WATER-sp) uses a 3-D grid of cells so that a processor that owns a cell only needs to look at neighboring cells to find interacting molecules. Communication arises out of the movement of molecules from one cell to another at every time-step.

4.2 Image Deblurring

The application tested is a parallel algorithm for deblurring of images obtained from Magnetic Resonance Imaging. The images generated may suffer a loss of clarity due to inhomogeneities in the magnetic field. One of the techniques for removing this blurring artifact is the demodulation of the data for each pixel of the image using the value of the magnetic field near that point in space. This method consists of acquiring a local field map, finding the best fit to a linear map and using it to deblur the image distortions due to local frequency variations. This is a very computation intensive operation and has previously been parallelized using a message passing approach [19]. Each thread deblurs the input image around its chosen frequency points in parallel. After deblurring around a frequency point, the relevant portions of the final image have to be updated. Since the portions updated by different threads could overlap, the update has to be done under the protection of a global lock.

4.3 Matrix Multiplication

The matrix multiplication program computes the product of two dense square matrices. The resultant matrix is divided across the processors using a row-wise block distribution. The size of each of the blocks has been set to a multiple of the page size of the machine. Each application thread computes at least one complete block of contiguous values. Hence there is no false sharing in this implementation and the application is close to ideal for execution on a page based DSM system.

4.4 Successive Over Relaxation

The successive over relaxation program (SOR) is a kernel for solving partial differential equations using a red-black algorithm. In every iteration, each point in a grid is set to the average of its four neighbors. Most of the traffic arises out of nearest neighborhood communication at the borders of the grids.

4.5 Program Parameters

The problem sizes used in this evaluation are shown in Table 2.

Program	Parameters
FFT	1048576 complex doubles
LU-c	2048 $\times$ 2048 doubles, block size 128
LU-n	1024 $\times$ 1024 doubles, block size 32
RADIX	1048576 integers, radix 1024
OCEAN-c	258 $\times$ 258 grid
WATER-n2	4096 molecules, 3 steps
WATER-sp	1000 molecules, 3 steps
MATMULT	1024 $\times$ 1024 doubles, 16 blocks
SOR	2002 $\times$ 1002 doubles, 100 iterations
MRI	14 frequency points, PHANTOM image

Table 2: Problem Sizes for Network Evaluation

All the test programs were run with different numbers of threads and processors as shown in Table 3.

Total tasks	Nodes Used	Threads per Node
1	1	1
2	2	1
4	4	1
8	4	2
16	4	4

Table 3: Task Distribution

The task distribution in case of two and four tasks was designed to exercise the network hardware. On an SMP cluster better results could have been obtained by using multiple application threads on a single node.

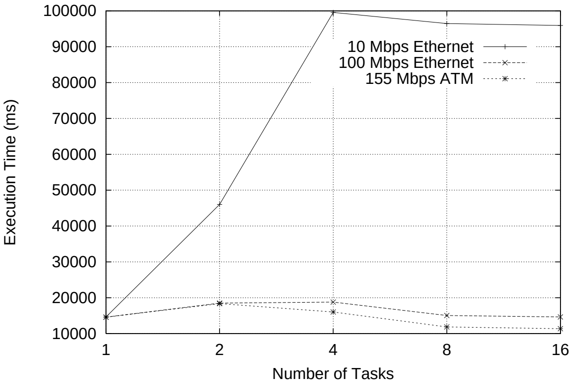

Figure 3: FFT

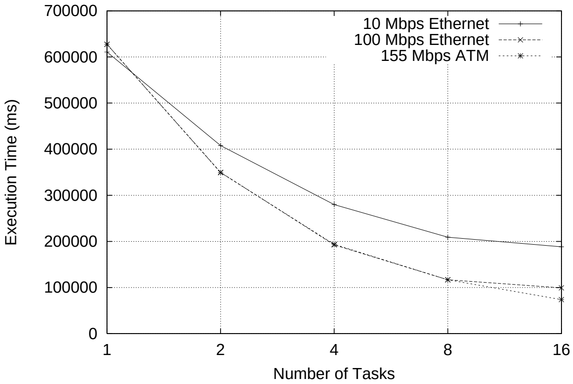

Figure 4: LU-c

5 Experimental Results

Table 4 shows the utilization of each network in terms of the the bandwidth used, and the average roundtrip latency per node. The average application packet size is the ratio of the total bytes transmitted to the number of messages sent. Both these numbers are characteristic of the application itself, and independent of the particular type of network. The bandwidth utilization (BW) was calculated as the ratio of the total bytes transmitted per task to the lifetime of the task. This is a conservative estimate, since some of the applications have a very high computation to communication ratio. The latency for a message includes the processing overhead at the remote node.

Program	Average Message Size (bytes)	10 Mbps Ethernet		100 Mbps Ethernet		155 Mbps ATM	
		BW (Mbps)	Latency (ms)	BW (Mbps)	Latency (ms)	BW (Mbps)	Latency (ms)
FFT	4945.50	2.12	134.23	12.71	174.16	15.84	111.68
LU-c	5976.38	1.93	431.08	3.71	192.73	4.68	256.80
LU-n	6223.95	2.02	178.30	9.32	146.84	9.87	148.68
RADIX	4886.12	2.16	32.31	12.85	3.62	14.83	2.80
OCEAN-c	4059.26	1.91	274.97	8.44	36.39	8.42	31.90
WATER-n2	197.19	0.62	118.15	0.77	170.82	0.71	313.30
WATER-sp	893.56	0.92	15.36	1.47	4.69	1.29	5.27
MATMULT	4385.72	1.00	48.57	1.42	7.03	1.45	5.24
SOR	7442.80	2.14	84.69	15.44	327.80	14.41	373.75
MRI	3440.01	2.13	7.31	4.39	1.70	4.40	1.62

Table 4: Communication Utilization Per Node

Programs like LU-c, WATER-n2, WATER-sp, MATMULT, and MRI have low bandwidth requirements and that is reflected in the small performance improvements while going from 10 Mbps Ethernet to the

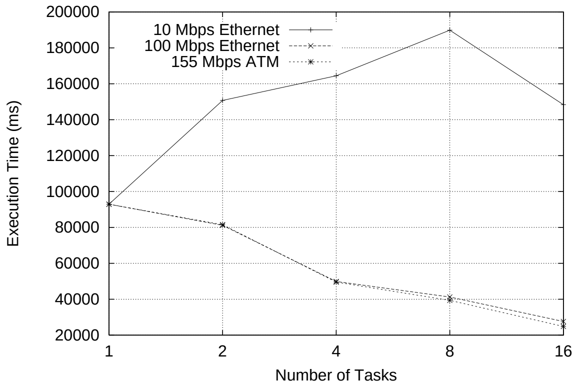

Figure 5: LU-n

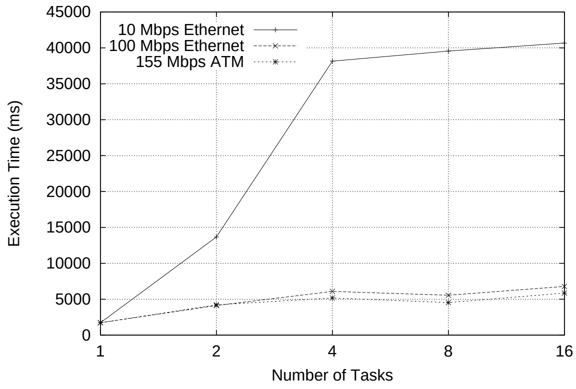

Figure 6: RADIX

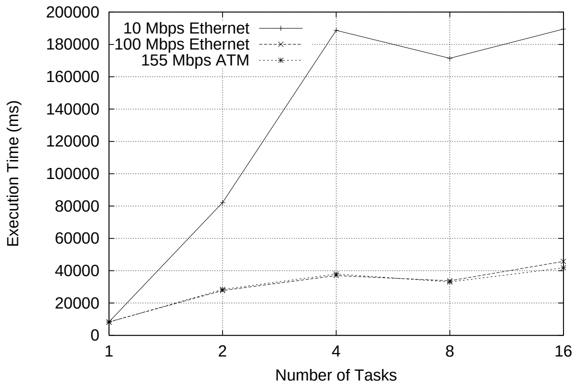

Figure 7: OCEAN-c

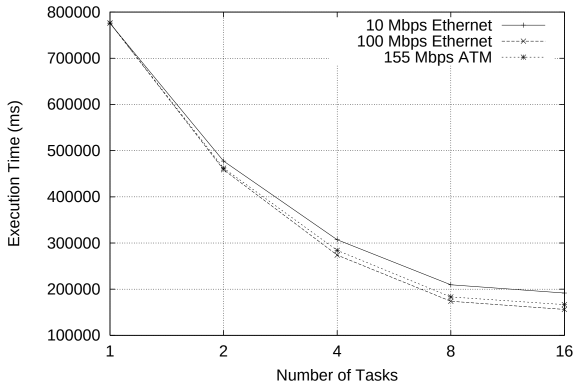

Figure 8: WATER-n2

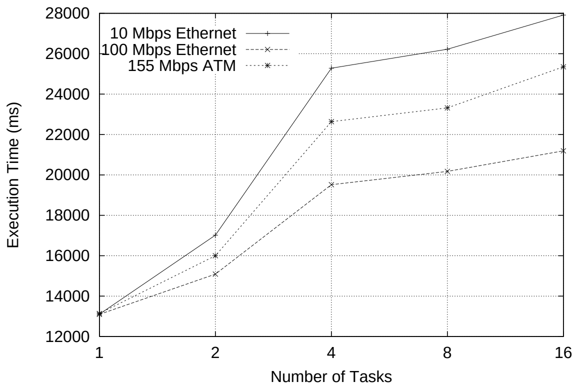

Figure 9: WATER-sp

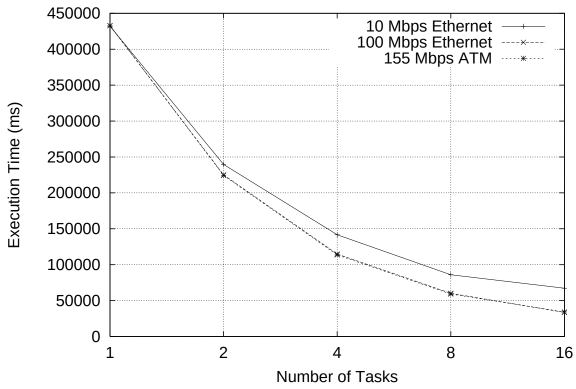

Figure 10: MATMULT

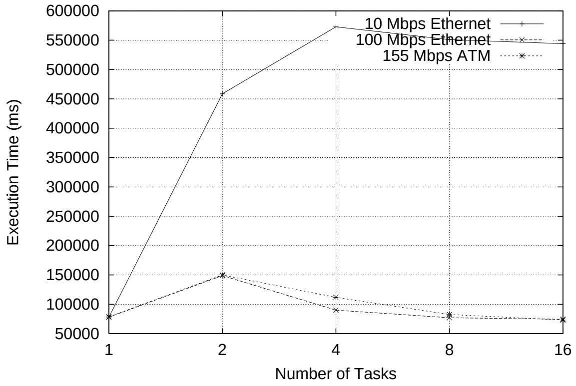

Figure 11: SOR

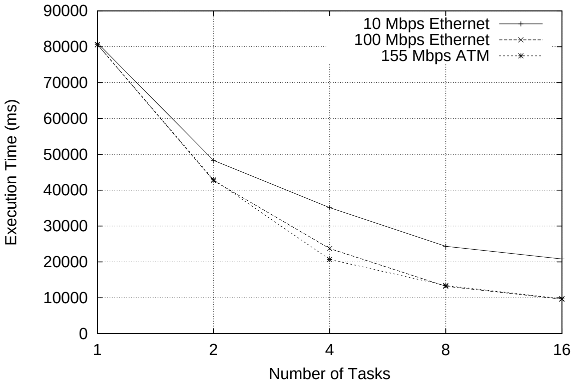

Figure 12: MRI

higher bandwidth networks as shown in Figures 4, 8, 9, 10, and 12. FFT, LU-n, and SOR have a high bandwidth requirement, and there is a distinct network bottleneck when using 10 Mbps Ethernet. This can be seen in Figures 3, 5, and 11, 100 Mbps Ethernet and 155 Mbps ATM are able to provide a speed-up when more nodes are added to the cluster, while the 10 Mbps cluster shows a slowdown with these programs. RADIX and OCEAN-c show similar trends in Figures 6 and 7. FFT, LU-c, LU-n, and RADIX have a large average packet size. From Figure 1, the latency of 100 Mbps Ethernet is lower than that of the ATM network only for packets less than approximately 4000 bytes. Hence one can see a some performance improvement by using the ATM network in these cases. On the other hand, WATER-n2 and WATER-sp have a very small packet size, and in this case the low latency 100 Mbps Ethernet outperforms the ATM network. For the other programs there is not much difference in the performance of the two networks.

MRI has a very high computation to communication ratio, and the final result is updated under the protection of a lock, hence it shows the least latency in all three cases. MATMULT has similar behavior, but at termination all threads try to update their part of the resultant matrix, which causes the latency to increase due to collisions at the receiving node.

Figure 13 shows the roundtrip times for *Strings* messages on each network, using different payload sizes. These results show that the communication system introduces a fair amount of latency when compared to the raw UDP data. The roundtrip time is measured for request-reply pairs, with no event handling at the server side. Hence these represent upper bounds that the current runtime system may achieve. This indicates that some work needs to be done to improve the performance of the *Strings* communication system

6 Conclusion and Future Work

Overall the communication results show that for five out of ten applications, namely LU-c, WATER-n2, WATER-sp, MATMULT, and MRI, the bandwidth utilization is not very high, and there does not seem to be much of an advantage from using higher bandwidth networks. However, for applications like FFT, LU-n, and SOR, the 10 Mbps Ethernet network poses a serious bottleneck. These programs show a speed-up when run on higher bandwidth networks, but a slowdown on 10 Mbps Ethernet.

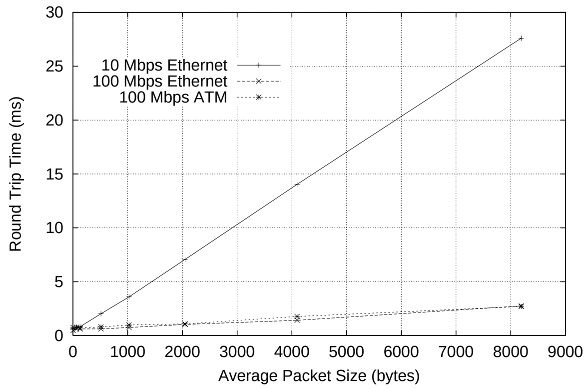

Figure 13: Roundtrip times for *Strings* messages

In the worst case, if all nodes communicate at the same time, on the same link, the total bandwidth requirement is still within the theoretical limits for both high bandwidth networks. However the latencies observed for most programs are much higher than predicted by the raw UDP data. This indicates that there is substantial software overhead in the runtime system.

The study also shows that in the current system, the extra cost of the ATM network and switch rarely translates into a performance improvement compared to the 100 Mbps Ethernet. Out future work would include studying the behavior of the programs when using switched FastEthernet, as well as switched Gigabit Ethernet.

Finally, the *Strings* communication subsystem introduces a large latency when compared to raw UDP packets. We plan to isolate these overheads more precisely, and investigate better approaches to providing a fast reliable communication system for the DSM.

References

[1] K. Li and P. Hudak, “Memory Coherence in Shared Virtual Memory Systems,” *ACM Transactions on Computer Systems*, vol. 7, pp. 321–359, November 1989.

[2] B. D. Fleisch and G. Popek, “Mirage: A Coherent Distributed Shared Memory Design,” in *Proceedings of the ACM Symposium on Operating System Principles*, (New York), pp. 211–223, ACM, 1989.

[3] J. Bennett, J. Carter, and W. Zwaenepoel, “Munin: Distributed Shared Memory Based on Type-Specific Memory Coherence,” in *Proceedings of the ACM Symposium on the Principles and Practice of Parallel Programming*, (New York), pp. 168–176, ACM, ACM Press, 1990.

[4] C. Amza, A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaeonopoel, “TreadMarks: Shared Memory Computing on Networks of Workstations,” *IEEE Computer*, pp. 18–28, February 1996.

[5] D. Khandekar, *Quarks: Portable Distributed Shared Memory on Unix*. Computer Systems Laboratory, University of Utah, beta ed., 1995.

[6] P. Keleher, *CVM: The Coherent Virtual Machine*. University of Maryland, CVM Version 2.0 ed., July 1997.

[7] S. Roy and V. Chaudhary, "*Strings:* A High-Performance Distributed Shared Memory for Symmetrical Multiprocessor Clusters," in *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, (Chicago, IL), pp. 90 –97, July 1998.

[8] S. C. Woo, M. Ohara, E. Torri, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Proceedings of the International Symposium on Computer Architecture*, pp. 24–36, June 1995.

[9] J. B. Carter, "Design of the Munin Distributed Shared Memory System," *Journal of Parallel and Distributed Computing*, 1995.

[10] R. Mirchandaney, S. Hiranandani, and A. Sethi, "Improving the Performance of DSM Systems via Compiler Involvement," in *Proceedings of Supercomputing 1994*, 1994.

[11] A. L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, and W. Zwaenepoel, "Software vs. Hardware Shared Memory Implementation: A Case Study," in *Proceedings of the International Symposium on Computer Architecture*, pp. 106 – 117, April 1994.

[12] C. Amza, A. Cox, K. Rajamani, and W. Zwaenepoel, "Tradeoffs Between False Sharing and Aggregation in Software Distributed Shared Memory," in *Proceedings of the ACM Symposium on the Principles and Practice of Parallel Programming*, (Las Vegas), pp. 90 – 99, June 1997.

[13] H. Lu, A. L. Cox, S. Dwarkadas, R. Rajamony, and W. Zwaenepol, "Compiler and Software Distributed Shared Memory Support for Irregular Application," in *Proceedings of the ACM Symposium on the Principles and Practice of Parallel Programming*, 1997.

[14] P. Keleher and C.-W. Tseng, "Enhancing Software DSM for Compiler-Parallelized Applications," in *Proceedings of International Parallel Processing Symposium*, August 1997.

[15] Information Networks Division, *Netperf: A Network Performance Benchmark*. Hewlett-Packard Company, http://www.netperf.org, revision 2.1 ed., February 1996.

[16] R. Hempel, H. C. Hoppe, U. Keller, and W. Krotz, *PARMACS V6.0 Specification. Interface Description*. Pallas GmbH, November 1993.

[17] D. Jiang, H. Shan, and J. P. Singh, "Application Restructuring and Performance Portability on Shared Virtual Memory and Hardware-Coherent Multiprocessors," in *Proceedings of the ACM Symposium on the Principles and Practice of Parallel Programming*, (Las Vegas), pp. 217 – 229, ACM, 1997.

[18] Y. Zhou, L. Iftode, J. P. Singh, K. Li, B. R. Toonen, I. Schoinas, M. D. Hill, and D. A. Wood, "Relaxed Consistency and Coherence Granularity in DSM Systems: A Performance Evaluation," in *Proceedings of the ACM Symposium on the Principles and Practice of Parallel Programming*, (Las Vegas), pp. 193 – 205, June 1997.

[19] P. Menon, V. Chaudhary, and J. G. Pipe, "Parallel Algorithms for deblurring MR images," in *Proceedings of ISCA 13th International Conference on Computers and Their Applications*, March 1998.