COMPaS: A Pentium Pro PC-based SMP Cluster and its Experience

Yoshio Tanaka[1], Motohiko Matsuda[1], Makoto Ando[1], Kazuto Kubota and Mitsuhisa Sato[1]

Real World Computing Partnership
{yoshio,matu,ando,kazuto,msato}@trc.rwcp.or.jp
1-6-1, Takezono, Tsukuba, Ibaraki 305, Japan

Abstract. We have built an eight node SMP cluster called COMPaS (Cluster Of Multi-Processor Systems), each node of which is a quad-processor Pentium Pro PC. We have designed and implemented a remote memory based user-level communication layer which provides low-overhead and high bandwidth using Myrinet. We designed a hybrid programming model in order to take advantage of locality in each SMP node. Intra-node computations utilize a multi-threaded programming style (Solaris threads) and inter-node programming is based on message passing and remote memory operations. In this paper we report on this hybrid shared memory/distributed memory programming on COMPaS and its preliminary evaluation. The performance of COMPaS is affected by data size and access patterns, and the proportion of inter-node communication. If the data size is small enough to all fit on the cache, parallel efficiency exceeds 1.0 using the hybrid programming model on COMPaS. But the performance is limited by the low memory bus bandwidth of PC-based SMP nodes for some memory intensive workloads.

1 Introduction

The cost/performance advantages of Pentium Pro processor systems have recently attracted widespread attention. Many researchers have reported on the design, implementation, administration, and performance analysis of Pentium Pro clusters[1]. Clusters using fast networks, such as Myricom's Myrinet[2], can provide high performance equal to MPPs[3]. **Symmetric multi-processor systems (SMPs)** are also becoming widespread, both as compute servers and as platforms for high performance parallel computing.

We have built a cluster of SMPs, **COMPaS (Cluster Of Multi-Processor Systems)**, which consists of eight quad-processor Pentium Pro (200MHz) SMPs connected by both a Myrinet high-speed network and a 100Base-T Ethernet.

For communication between nodes, we consider remote memory operations to be more suitable than message passing on SMP clusters, because message passing suffers from the handling of incoming messages. Message passing operations need mutual exclusions on buffers and message copying burdens the limited bus bandwidth. To overcome these problems, we designed and implemented a user-level communication layer for Myrinet, **NICAM**. NICAM provides high-bandwidth and low-overhead remote memory transfers and synchronization primitives. NICAM transfers data using the direct memory access (DMA) engine on the Myrinet Network Interface (NI) which utilizes not the central processor but the microprocessor on the NI. NICAM is implemented by an Active Message (AM) mechanism[6] on the Myrinet NI. In message passing, sender and receiver synchronize implicitly when they send/receive messages, however, in remote memory operations, each node writes to remote memory asynchronously. Therefore, since all nodes need to synchronize explicitly for phasing computations, fast synchronization primitives should be provided. Active Messages on NI enable fast barrier synchronization by reducing the host-NI overhead.

Architectures of parallel systems are broadly divided into two categories: shared memory and distributed memory. While multi-threaded programming is used for parallelism on shared memory systems, the typical programming model on distributed memory systems is message passing. SMP clusters are considered to be a mixed configuration of shared memory and distributed memory. To achieve high performance on SMP clusters, we need to take advantage of both architectures. An SMP cluster may be a good choice for building cluster systems but there are many unknown programming factors and performance characteristics.

In this paper, we present a hybrid shared memory/distributed memory programming model on COMPaS. Processors within each node exchange data through shared memory. Processors on different nodes communicate through 100Base-T Ethernet or Myrinet. We measured the performance of COMPaS using various workloads including a Laplace equation solver, matrix-matrix multiplication, Conjugate Gradient Kernel, and a Radix Sort. The Message Passing Interface (mpich-1.1.0 on TCP/IP)[4, 5] can be used for communications through 100Base-T Ethernet. The experimental results for 100Base-T Ethernet provide researchers with useful information about the performance of Pentium Pro PC-based SMP clusters having a standard configuration. We also used NICAM for communications through Myrinet. The experimental results for Myrinet gave the highest performance for Pentium Pro PC-based SMP clusters.

There are many research efforts in cluster computing. The Illinois High Performance Virtual Machines (HPVM) Project[9] researches software technology for scalable clusters. Configuration of HPVM Pentium Pro cluster includes 64 nodes with a total of over 112 processors, 6GB memory, and 256KB disk. The interconnects of the cluster include Myrinet, Compaq/Tandem's Servernet, and 100 Megabit Ethernet. The Berkeley NOW project[10] is building system support for using a network of workstations (NOW). The NOW project has been exploring larger clusters with high-speed networks. The cluster of NOW project is including SUN Ultrasparcs, SUN Sparcstations, Intel PCs. The Jazznet[11] is a cluster of Linux PCs used for applied math and scientific computations. The original Jazznet configuration consisted of a mixture of single and multi-processor Pentium Pro PCs: three single-processor machines, one dual-processor, and one quad-processor system. The Jazznet provides environments for experiments with both distributed and shared memory applications.

Although many PC-based cluster systems are reported beside these cluster systems, no homogeneous SMP cluster systems like our COMPaS and hybrid programming on the system have been reported.

In this paper, we report on the design and the performance of NICAM, a hybrid shared memory/distributed memory programming model on COMPaS and its preliminary evaluation, and the performance characteristics of COMPaS. We describe the specifications and basic performance of COMPaS in the next section. Section 3 describes the design and the performance of NICAM. The hybrid programming model is presented in section 4. The experimental results and the performance of COMPaS are shown in section 5. In section 6, we discuss the performance characteristics of COMPaS. Our summary and conclusion are in section 7.

2 Configuration and Performance of Node Processor

In this section, we present the configuration of COMPaS and the performance of node processor. Figure 1 illustrates the configuration of COMPaS. COMPaS consists of eight quad-processor Pentium Pro PC servers (Toshiba GS700, 450GX chip-set, 200MHz, 16KB L1 cache, 512KB L2 cache, 128MB Main Memory) connected to both Myrinet and 100Base-T Ethernet switches. The operating system on each node is Solaris 2.5.1.

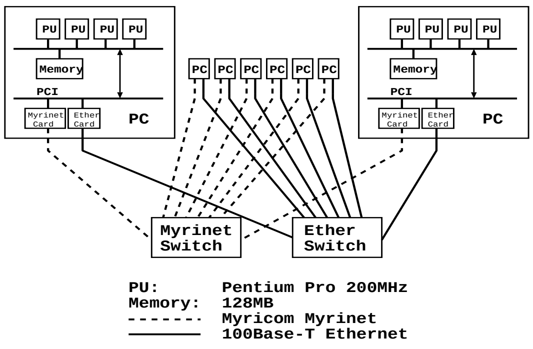

Fig. 1. Configuration of COMPaS

2.1 Memory Bus Bandwidth

Table 1 shows the memory bus bandwidth for memory read, memory write, and memory copy. Table 2 shows the *bcopy* bandwidth when multiple threads execute *bcopy* operations simultaneously.

Here, the second column shows the average bandwidth of the threads. The third column shows the total bandwidth of all threads. Note: the total bandwidth of all threads does not depend on the number of threads, i.e. the *bcopy* bandwidth per thread is limited by the total *bcopy* bandwidth which is about 74MB/s.

Table 1. Memory bus bandwidth

read(MB/s)	write(MB/s)	copy(MB/s)
203.5	97.9	70.5

Table 2. *Bcopy* bandwidth with multiple threads

Threads	Bandwidth(MB/s)	Total Bandwidth(MB/s)
1	71.31	71.31
2	37.06	74.12
3	24.75	74.25
4	18.56	74.24

2.2 Synchronization Time between Threads

Table 3 shows the synchronization time between threads. Our algorithm uses a spin lock, and does not use any mutex or condition variables provided by the operating system. Our algorithm provides very fast barrier synchronization, and takes less than 2 microseconds for four threads. The synchronization time using Solaris operating system mutex variables is about 180 microseconds for four threads.

Table 3. Sync. time between threads

Num. of Threads	2	3	4
Sync. Time (μsec)	1.222	1.761	1.960

3 NICAM: User-level Communication Layer for Myrinet

In this section, we describe an outline of NICAM and its performance. Remote memory operations are considered more suitable than message passing, because remote memory operations eliminate handling of message buffers. We designed and implemented a user-level communication layer for Myrinet which provides low-overhead and high-bandwidth communication[7]. In this section, we describe the design and the performance of NICAM.

3.1 Communication Primitives Using Active Messages

For inter-node communications on the one hand, communications latency affects the performance of the communications but on the other hand communication host CPU overhead is considered more important in SMP clusters because it occupies the bus and affects other processors on the same node. We provide remote memory data transfer primitives and barrier synchronization operations by using Active Messages on the NI. Remote memory based operations require no handling of message buffers in the lower communication layers on for flow control. They are implemented by the Active Message mechanism on Myrinet NIs to enable low-overhead and high-bandwidth DMA without extra message copying. We also used Active Messages for requests from the main processor to the Myrinet NI. Active Messages exchanged between NIs directly invoke DMA on the remote NIs without involving a host processor.

3.2 Performance of NICAM

Figure 2 shows the bandwidth and the communication time of NICAM. The minimum latency for small messages is about 16 microseconds and the maximum bandwidth is about 82.5 MB/s. Table 4 compares the synchronization time between nodes by NICAM and PM[3] on COMPaS. PM is also a user level message passing library for Myrinet. We used synchronization primitives for NICAM. We synchronized all the nodes with point-to-point communications combined using a shuffle exchange algorithm for PM. Although the synchronization time for two nodes by NICAM is larger than PM because of the host-NI communication cost, the communication time taken for each step for NICAM is about 7.5 microseconds, which is less than half of the time for PM (17 microseconds). In our COMPaS, MPICH is available as a standard communication layer over Ethernet. Figure 3 shows the bandwidth of point-to-point communication using *MPI_Send()* and *MPI_Recv()* functions. The maximum bandwidth is about 4MB/s. Table 5 shows the synchronization time between nodes using *MPI_Barrier()*. NICAM allows considerably higher performance of data transfers and synchronizations between nodes than MPICH with 100Base-T Ethernet.

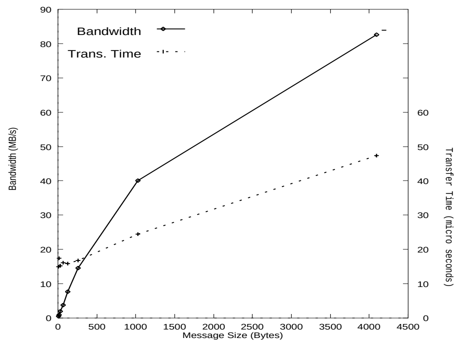

Fig. 2. Performance of remote memory based data transfer

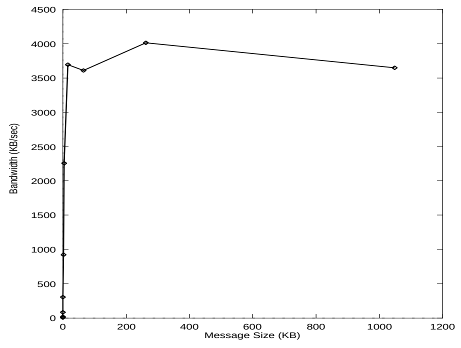

Fig. 3. Point-to-point communication bandwidth (MPICH over 100Base-T Ethernet)

Table 4. Barrier sync. time between nodes using NICAM on Myrinet

Num. of Nodes	2	4	8
NICAM(μsec)	23.79	31.45	38.67
PM(μsec)	13.47	30.37	47.42

Table 5. Barrier sync. time between nodes using MPICH on 100Base-T

Num. of Nodes	2	4	8
Sync. Time (μsec)	493	1066	1645

4 Hybrid Shared Memory/Distributed Memory Programming

We designed a hybrid programming model in order to take advantage of locality in each SMP node. Intra-node computations utilize a multi-threaded programming style (Solaris threads) and inter-node programming is based on message passing and remote memory operations. In this paper, we will focus on data parallel programs. In this section, we will outline the hybrid programming model. Fig. 4 illustrates our hybrid programming model.

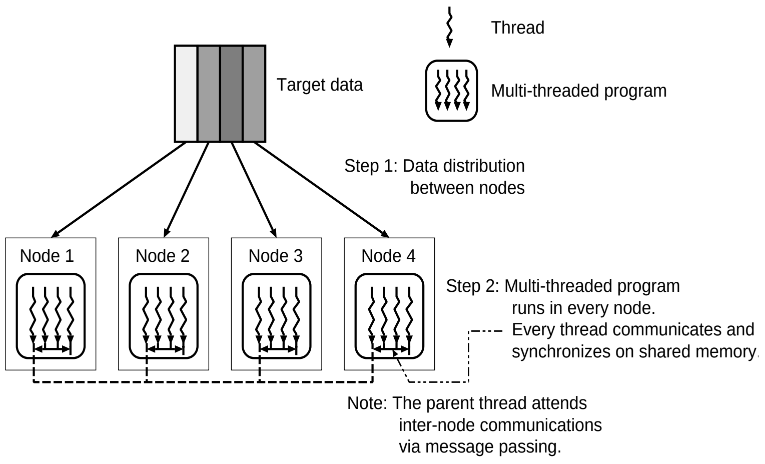
Thread
Multi-threaded program
Target data
Step 1: Data distribution between nodes
Node 1
Node 2
Node 3
Node 4
Step 2: Multi-threaded program runs in every node.
Every thread communicates and synchronizes on shared memory.
Note: The parent thread attends inter-node communications via message passing.

Fig. 4. Hybrid programming model

In data parallel programs, we can phase the partitioning of target data such as matrices and vectors easily. First, we partition and distribute the data between nodes and then partition and assign the distributed data to the threads in each node. Therefore, it is easy to implement data parallel programs based on the hybrid programming model for COMPaS. The hybrid programming model is based on SPMD programming style. Data decomposition and distribution method and inter-node communications are the same as the distributed programming strategy. Data allocation to threads and local computation are the same as a multi-threaded programming on shared memory systems. Hybrid programming is considered as a type of distributed programming such that multiple threads are used for the computation in each node. Although some operations such as *reduction* and *scan* need more complicated steps in hybrid programming, we can easily implement hybrid programming by combining both shared and distributed programming.

5 Experimental Results

We implemented some workloads, including a Laplace equation solver, matrix-matrix multiplication, Conjugate Gradient Kernel from the NAS Parallel Benchmarks, and a Radix Sort. We measured the execution time of the four benchmarks on COMPaS. The matrix size of the Laplace equation solver is 640×640. The matrix size of the matrix-matrix multiplication is 1800×1800. The size of the CG Kernel is class A. In Radix Sort, we sorted 4M 32 bit integers. We varied the number of nodes by using 1, 2, 4, or 8. In each case, we also varied the number of threads by using 1, 2, or 4 threads. The results for one node represent that of the multi-threaded programming version on shared memory systems. The results for one thread indicate the results of the message passing / remote memory based programming version of the benchmarks on distributed memory systems.

Explicit Laplace Equation Solver
Figures 5 and 6 show the execution time of the Explicit Laplace equation solver. Figure 5 shows the results done through Myrinet using NICAM. Figure 6 shows the results done through 100Base-T using MPICH.

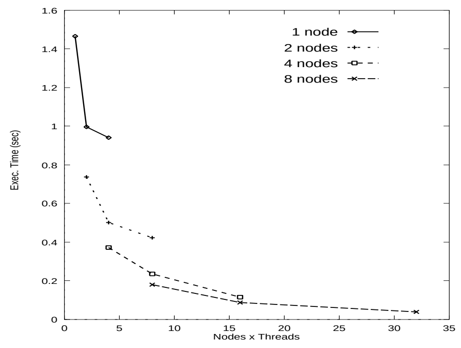

Fig. 5. Exec. time of Laplace equation solver (N=640, Hybrid, NICAM, Myrinet)

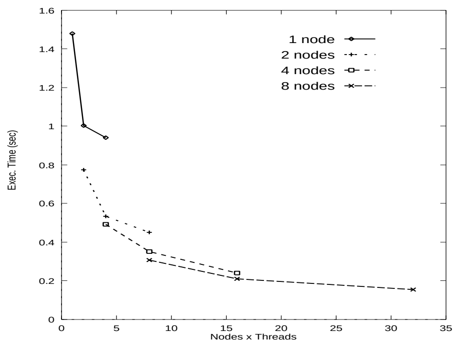

Fig. 6. Exec. time of Laplace equation solver (N=640, Hybrid, MPICH, 100Base-T)

Each line represents the execution time for 1 node, 2 nodes, 4 nodes, and 8 nodes respectively. Horizontal coordinates are the product of the number of nodes and the number of threads in each node, that is, the total number of threads.

The results for one node (meaning an ordinary multi-threaded version) show very low scalability for the number of threads because the performance of the memory bus becomes bottleneck since the data size is so large that access to main memory occurs frequently. As the number of nodes increases, the amount of local computation decreases, but the message size of inter-node communication does not depend on the number of nodes. Although all inter-node communications on the Explicit Laplace equation solver are between neighbor processors, the experimental results reflect the performance of inter-node communications clearly. The execution time done through Myrinet using NICAM is less than half of one done through 100Base-T using MPICH. Figures 7 shows the speedup of the Explicit Laplace equation solver done through Myrinet using NICAM. Horizontal coordinates are the same as Figure 5. *ideal* shows the ideal speedup.

The speedup for the eight node case exceeds the ideal speedup because the working set becomes small enough to fit into the cache and NICAM provides

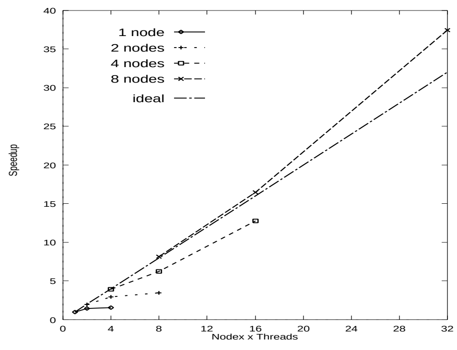

Fig. 7. Speedup of Laplace equation solver (N=640, Hybrid, NICAM, Myrinet)

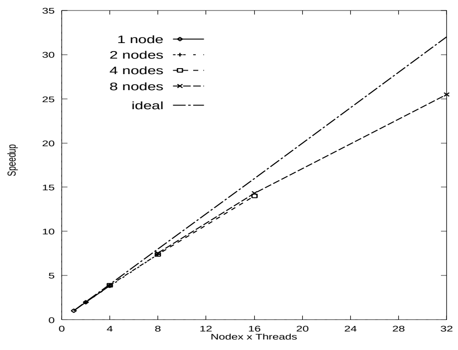

Fig. 8. Speedup of matrix-matrix multiplication (N=1800, Hybrid, NICAM, Myrinet)

fast inter-node communications.

Matrix-matrix Multiplication
Figure 8 shows the speedup of the matrix-matrix multiplication done through Myrinet using NICAM. X coordinates are the same as Figure 5. *ideal* shows the ideal speedup.
Although the matrix size is too large to fit into the cache, our blocking and tiling algorithm which use the cache effectively can provide high performance for the single node case. The efficiency for the 8 node and 4 thread case is about 80%. The combination of high data locality by using cache effectively and high performance inter-node communications enables high performance with hybrid programming.

CG Kernel
Figure 9 shows the speedup of the CG Kernel done through Myrinet using NICAM. X coordinates are the same as Figure 5. *ideal* shows the ideal speedup.

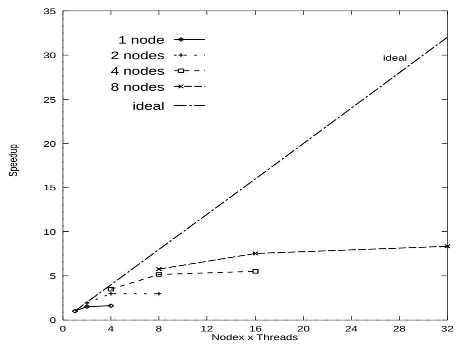

Fig. 9. Speedup of CG Kernel (Class A, Hybrid, NICAM, Myrinet)

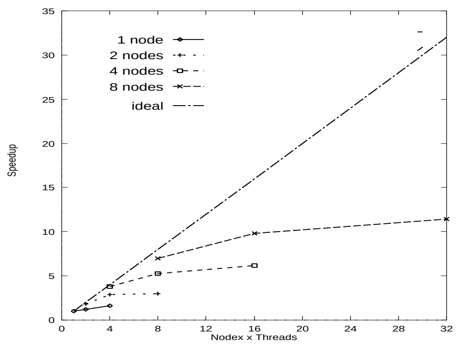

Fig. 10. Speedup of Radix Sort (N=4M int, Hybrid, NICAM, Myrinet)

The results for one node show very low scalability for the number of threads. The execution time with four threads is almost the same as the time for two threads. Because the data size of CG Kernel (class A) is very large and accessing the main memory occurs frequently, the performance of the memory bus becomes a bottleneck.

Radix Sort
Figure 10 shows the speedup of Radix Sort done through Myrinet using NICAM. X coordinates are the same as Figure 5. *ideal* shows the ideal speedup.
The results for one thread (meaning an ordinary distributed programming version) indicate a reasonable scalability for the number of nodes. But the results for one node show very low scalability for the number of threads. The results for CG Kernel and Radix Sort are not satisfactory when compared to the results for Laplace equation solver because their data exceeds cache capacity and they run into the memory bus bottleneck.

6 Performance Characteristics of COMPaS

6.1 Advantage of Locality

Memory intensive workloads such as CG Kernel and Radix Sort are difficult applications for SMP clusters with low memory bus bandwidth. The data size and the access pattern to the data are considered as important factors which affect whether we can take advantage of locality or not. If the data size is small enough to store all of the data on the cache, we can take advantage of locality in each SMP node and get high performance by hybrid programming. The results of Laplace equation solver are interesting. The results for 1 node do not present good scalability for multi-threads because the data size for 1 node is not small enough to store all of the data on the cache, however the results for 8 nodes indicates high scalability for multi-threads because the data size is reduced and now we can take the advantage of locality in each SMP node. Figure 11 shows the execution time of CG Kernel (class Tiny) based on the shared memory programming model.

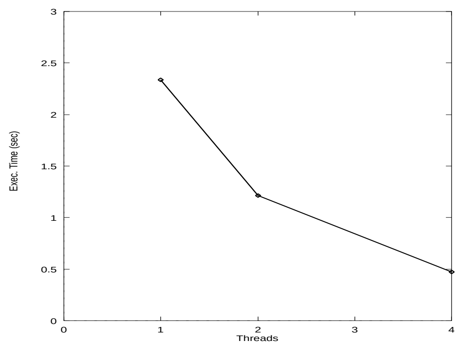

Fig. 11. Exec. time of CG Kernel (Class Tiny, Multi-Threads)

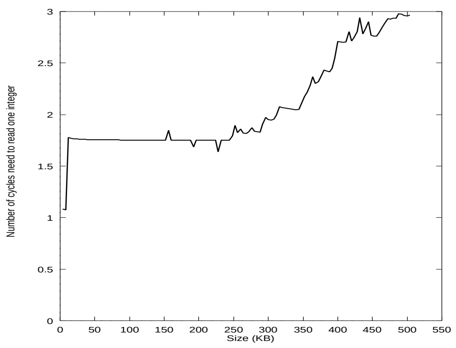

Fig. 12. The number of cycles to read one integer

The matrix size of class Tiny is 180,000 doubles, that is about 1MB. If we invoke two threads, the matrix data size for each thread is about 500KB and cache misses occur when reading the matrix. If however, we invoke four threads, the matrix data size for each thread is about 250KB and it is now possible to store all of the data on the cache. The efficiency for two threads is about 0.96, but the rate for four threads exceeds 1.0 and is about 1.24. Because Laplace equation solver has a small working set and its inter-node communication is not all-to-all but only with its neighbors, its results are better than CG Kernel and Radix Sort.

The data access pattern is an another important factor to take advantage of locality. If processors access data on the cache frequently, that means the ratio of the memory accesses to the number of operations on the data in the cache is low, we can take advantage of the locality even if the data size is large. For example, in matrix-matrix multiplication, if a good blocking algorithm can be applied and the data on the cache are frequently accessed by processors, we can get high performance on COMPaS (see Figure 8).

6.2 Memory Bus Bottleneck

In our hybrid programming model, COMPaS can not provide high performance for applications with large working sets including the CG Kernel and Radix Sort. Especially, the CG Kernel has large data sets, and the access pattern to the sparse matrix data is like scanning, that is, each element of the matrix is referred to only once in each matrix-vector multiplication. As a result, there are frequent accesses to the main memory.

We analyzed the code of the matrix-vector multiplication of the CG Kernel and made a rough estimate of the peak performance memory bus bandwidth required by each thread to read the matrix. It is about 120MB/s. As shown in Table 1, the memory bus bandwidth for memory read is 208MB/s and when multiple threads are invoked, the bandwidth for each thread is limited by that value. If we invoke four threads for the CG Kernel, each thread requires 120MB/s bandwidth for matrix-vector multiplication, however all threads actually acquire 50MB/s of bandwidth on average. The results for CG Kernel and Radix Sort show that for any number of nodes, the execution time for 2 threads and 4 threads is almost the same. This is because the memory bus bandwidth becomes a bottleneck.

6.3 Effect of Inter-node Communication Performance

Table 6 shows the efficiency of matrix-matrix multiplication done through Myrinet using NICAM. N is the number of nodes and T is the number of threads.

Table 6. Efficiency of matrix-matrix multiplication (N=1800, Hybrid, AM, Myrinet)

	T = 1	T = 2	T = 4
N = 1	1.00	0.99	0.97
N = 8	0.94	0.89	0.80

Although in the single node case, we can get high scalability for the number of threads, the scalability for the number of threads is not satisfactory in the 8 node case. We broke down the execution time for the 8 node case. The time for local computation (sub-matrix multiplication done by threads) is 17.09 sec, 8.59 sec, and 4.46 sec for 1, 2, 4 threads respectively. This means we can get high scalability on the local computation. But the time for inter-node communications does not depend on the number of threads, and it is 0.83 sec. As the number of threads increases, the inter-node communication time is revealed more clearly.

6.4 Effective Cache Size

As described in section 2, the size of the L2 cache is 512KB. But we can't get high performance on Laplace equation solver even if the data size for each thread is less than 500KB. We get high performance when the data size for each thread is less than about 200KB. We explored the reasons why we can't take advantage of multi-threads in the *matrix-vector multiplication* and performed an experiment to examine the effect of the cache. We read *size* bytes as integers from the main memory and get the number of cycles taken for reading *size* data using performance counter register(s) of the Pentium Pro.

Figure 12 shows the average cycles need to read one integer. Although the size of the L2 cache is 512KB, it is a physical address cache and cache misses begin to occur when the data size exceeds about 200KB because of the cache line conflict. In the worst case, cache misses first occurred when reading 160KB data. If the data size for each thread exceeds about 200KB, cache misses and main memory accesses occur, and the performance of the memory bus becomes a bottleneck. This is the reason why high scalability is unobtainable for the number of threads in Laplace equation solver.

7 Summary and Conclusion

We have built the COMPaS cluster of SMPs, which consists of eight quad-processor Pentium Pros. In this paper, we reported the basic performance of COMPaS and a hybrid shared memory/distributed memory programming model and its preliminary evaluation. We also described the NICAM user-level communication layer. NICAM is based on Active Messages and on the Myrinet Network Interface and provides low-overhead and high-bandwidth.

For the solution of the Laplace equation, we obtain high performance if the Jacobi method is used, however the performance of the CG Kernel is not satisfactory because of the memory bus bottleneck and the global communications bottleneck. Here are some guidelines when programming PC-based SMP clusters:

– If multiple threads are invoked on each node and the data size for each thread is very large, memory bus performance may limit the performance of the SMP cluster system.

- In SMP clusters, a multi-thread safe implementation of message passing libraries such as MPI is necessary. To circumvent the bottleneck caused by the bus system, communication primitives which allow direct access to remote memory are desirable. We implemented such primitives by using Active Message and DMA mechanisms on the Myrinet NI.
- If the data size is small enough to fit into the cache, we can take the full advantage of multiple threads in each SMP node.
- To achieve high performance on a Pentium Pro PC-based SMP cluster, it is important to exploit locality, meaning reduce the ratio of the memory accesses to the number of operations on the data in the cache. If this ratio is low then PC-based SMP clusters can provide high performance even if the data size is large.

In order to tolerate inter-node communication, we are now investigating a programming scheme which overlaps communication and computation. Our experimental results show the inter-node communication time is revealed clearly when many threads is running. In such cases, overlapping programming is expected to provide high performance.

References

1. "Pentium Pro Cluster Workshop", http://www.scl.ameslab.gov/workshops/.
2. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic and S. Wen-King, "Myrinet – A Gigabit-per-Second Local-Area Network," IEEE MICRO, Vol. 15, No. 1, pp. 29–36 (1996).
3. H. Tezuka, A. Hori, Y. Ishikawa and M. Sato, "PM: An Operating System Coordinated High Performance Communication Library", High-Performance Computing and Networking, Lecture Notes in Computer Science, Vol. 1225, pp. 708–717, Springer-Verlag (1997).
4. "The Message Passing Interface (MPI) standard", http://www.mcs.anl.gov/mpi/index.html.
5. W. Gropp and E. Lusk, "MPICH Working Note: Creating a new MPICH device using the Channel interface", Argonne National Laboratory Technical Report (1995).
6. T.von Eicken, D.E.Culler, S.C.Goldstein and K.E.Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation", Proc. 19th Int'l Symp. on Computer Architecture, pp. 256-266 (1992).
7. M. Matsuda, H. Tezuka, Y. Tanaka, K. Kubota, M. Ando and M. Sato, "Network Interface Active Messages on SMP Clusters" Proc. IPSJ SIGARC, Vol. 97, No. 76, pp. 55–60 (1997) (in Japanese).
8. Y. Kodama, H. Sakane, H. Koike, M. Sato, S. Sakai, and Y. Yamaguchi, "Parallel Execution of Radix Sort Program Using Fine-Grain Communication", Proc. PACT'97, pp. 136-145 (1997).
9. "Scalable Clusters of Commodity Computers", http://www-csag.cs.uiuc.edu/projects/clusters.html.
10. Thomas E. ANderson, David E. Culler, David A. Patterson and the NOW Team. "A Case for Networks of Workstations: NOW", IEEE Micro (1995).
11. "The Jazznet Project", http://math.nist.gov/jazznet/index.html.

This article was processed using the LaTeX macro package with LLNCS style