Recently, symmetric multiprocessor systems have become widely available, both as computational servers and as platforms for high-performance parallel computing. The same trend is found in PCs—some have also become SMPs with many CPUs in one box. Clusters of PC-based SMPs are expected to be compact, cost-effective parallel-computing platforms. At the Real World Computing Partnership, we have built a PC-based SMP cluster, COMPaS (Cluster of Multi-Processor Systems), as the next-generation cluster-computing platform (see the sidebar). Our objective is to study the performance characteristics of a PC-based SMP cluster and the programming models and methods for an SMP cluster.

There have been many other research efforts in cluster computing, including the Illinois High Performance Virtual Machines project (http://www.csag.cs.uiuc.edu/projects/clusters.html), the Berkeley NOW project,2 and Jazznet (http://math.nist.gov/jazznet/index.html). However, few researchers have studied a homogeneous, medium-scale PC-based SMP cluster like our COMPaS.

COMPaS
The cluster consists of eight quad-processor Pentium Pro SMPs connected to both Myrinet and 100Base-T Ethernet switches (see Figure 1). We chose Solaris 2.5.1 (x86) as the operating system on each node, because it provides a stable, mature environment for multithreaded programming on SMP nodes.

The memory bus bandwidth for the Pentium Pro SMP node is 203.5 Mbytes per second for read, 97.9 Mbytes per second for write, and 70.5 Mbytes per second for copy. Table 1 shows the bandwidth when multiple threads execute bcopy operations simultaneously. The total bandwidth of all threads does not depend on the number of threads—the bcopy bandwidth per thread is limited by the total bcopy bandwidth, which is approximately 74 Mbytes/s.

Our algorithm for barrier synchronization uses a spin lock and does not use any mutex or condition variables provided by the operating system. It provides very fast barrier synchronization and takes less than 2 microseconds for four threads, only 1.76 μs for three threads, and 1.22 μs for two threads. The synchronization time using Solaris mutex variables is approximately 180 μs for four threads.

NICAM: A USER-LEVEL COMMUNICATION LAYER
For communication between SMP cluster nodes, remote memory operations are more suitable than message passing, 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 solve these problems, we designed and implemented a NICAM4 (Network Interface Active Message) user-level communication layer using Myrinet.

NICAM provides high-bandwidth and low-overhead remote memory transfers and synchronization primitives. Overhead refers to processor involvement in data transfer. In multithreaded programming environments in SMP

**Table 1**

<table>
<thead>
<tr>
<th>Thread Count</th>
<th>Read Bandwidth (Mbytes/s)</th>
<th>Write Bandwidth (Mbytes/s)</th>
<th>Copy Bandwidth (Mbytes/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>203.5</td>
<td>97.9</td>
<td>70.5</td>
</tr>
<tr>
<td>2</td>
<td>101.75</td>
<td>48.95</td>
<td>35.25</td>
</tr>
<tr>
<td>3</td>
<td>67.87</td>
<td>32.63</td>
<td>23.5</td>
</tr>
<tr>
<td>4</td>
<td>52.17</td>
<td>23.88</td>
<td>17.7</td>
</tr>
</tbody>
</table>

**RWCP cluster-computing research**

The Real World Computing Partnership is a 10-year Japanese project started in 1992 and funded by the Japanese Ministry of International Trade and Industry. Those of us involved in the partnership pursue cluster technology for high-performance parallel and distributed computing.

We built the RWC PC Cluster I (32-node, 166-MHz Intel Pentiums) using a Myricom Myrinet Gbit network as our high-speed interconnection network and compact packaging with industrial standard PC cards. These PC clusters use a very efficient communication library, PM, using Myrinet to achieve a level of performance comparable to traditional MPPs. We have also built its successor, the RWC PC Cluster II (128-node, 200-MHz Intel Pentium Pros), to demonstrate our PC cluster’s scalability and computational power (http://www.rwcp.or.jp/lab/pdlab/).
clusters, a multithreaded safe implementation of message-passing libraries, such as MPI, is necessary. Message passing incurs overhead from coordination between processors in an SMP node and requires control of the message flow. The overhead includes managing message buffers and copying messages that burden the limited bus bandwidth.

In NICAM, we implemented remote memory-data-transfer primitives and barrier-synchronization operations by running an Active Messages mechanism on the Myrinet Network Interface (see Figure 2). We also used AMs for requests from the main processor to the Myrinet NI. AMs exchanged between NIs directly invoke direct memory access on the remote NIs without involving a host processor.

NICAM extensively uses a cache-coherence mechanism to synchronize with the message at the receiver side—that is, the receiver waits for the message to arrive. When an event occurs (such as a message arrives or is sent), the flag at the specified location in the main memory is set. The program knows of the event by checking the flag. A NICAM synchronization primitive sets a flag in the main memory when the message arrives. Processors waiting for the memory location do not generate bus traffic on the cache-coherent bus because the event is checked in the cache. For example, a NICAM remote memory-write operation, `nicam_bcopy_notify`, sends the data directly to the destination and sets the specified flag to indicate the message's arrival.

All communications necessary to implement barrier synchronization are performed between the NIs. Although the barrier uses a buffer-fly algorithm, there is no need for a main processor to poll incoming messages. This design makes barrier synchronization faster and reduces the overhead.

Figure 3 shows NICAM’s bandwidth and communication time. The minimum latency for small messages is approximately 20 μs (including time to receive its ack message), and the call overhead for data transfer is 5.7 μs. The maximum bandwidth is approximately 105 Mbytes/s and $N_{\text{half}}$ (the size at which the message-transfer bandwidth achieves half of the peak bandwidth) is 2 Kbytes.

Table 2 compares the synchronization time between nodes for NICAM and PM on COMPsaS. PM is also a user-level message-passing library for Myrinet. With NICAM, we used synchronization primitives. With PM, we synchronized the nodes with point-to-point communications that were combined through a shuffle exchange algorithm. Although the synchronization time for two nodes by NICAM is larger than PM because of the host-NI communication cost, the communication time for each step for NICAM is approximately 7.5 μs, which is less than half the time for PM (17 μs).

On COMPsaS, MPICH is available as a standard communication layer over Ethernet. For a comparison, Table 2 also shows the synchronization performance of `MPI_barrier` using MPICH. The maximum bandwidth of point-to-point communication using `MPI_Send()` and `MPI_Recv()` functions is approximately 4 Mbytes/s. NICAM allows considerably higher performance for data transfers and synchronization between nodes than does MPICH with 100Base-T Ethernet.

<table>
<thead>
<tr>
<th>THREADS</th>
<th>AVERAGE (MBYTES/S)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BANDWIDTH</td>
</tr>
<tr>
<td>1</td>
<td>71.31</td>
</tr>
<tr>
<td>2</td>
<td>37.06</td>
</tr>
<tr>
<td>3</td>
<td>24.75</td>
</tr>
<tr>
<td>4</td>
<td>18.56</td>
</tr>
</tbody>
</table>

Figure 1. COMPsaS consists of eight quad-processor Pentium Pro PC servers (Toshiba GS700, 450GX chip set, 16-Kbyte L1 cache and 512-Kbyte L2 cache).

Figure 2. Network Interface Active Message implementation.
PARALLEL PROGRAMMING FOR SMP CLUSTERS

Architectures of parallel systems are broadly divided into shared-memory and distributed-memory models. 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 have a mixed configuration of shared-memory and distributed-memory architectures. One way to program SMP clusters is to use an all-message-passing model. This approach uses message passing even for intranode communication. It simplifies parallel programming for SMP clusters but might lose the advantage of shared memory in an SMP node. Another way is with the all-shared-memory model, using a software distributed-shared-memory (DSM) system such as TreadMarks. This model, however, needs complicated runtime management to maintain consistency of the shared data between nodes.

We use a hybrid programming model of shared and distributed memory to take advantage of locality in each SMP node. Intranode computations use multithreaded programming, and internode programming is based on message passing and remote memory operations.

Consider data-parallel programs. We can easily phase the partitioning of target data such as matrices and vectors. First, we partition and distribute the data between nodes and then partition and assign the distributed data to the threads in each node. Data decomposition and distribution and internode communications are the same as in distributed-memory programming. Data allocation to the threads and local computation are the same as in multithreaded programming on shared-memory systems. Hybrid programming is a type of distributed programming, in that computation in each node uses multiple threads. Although some data-parallel 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 for data-parallel programs.

EXPERIMENTAL RESULTS

We measured the performance of COMPaS using various workloads, including a Laplace equation solver, a conjugate gradient kernel, matrix-matrix multiplication, and Cholesky factorization.

EXPLICIT LAPLACE EQUATION SOLVER

We use the Jacobi method to solve a Laplace equation on a $640 \times 640$ matrix. Figure 4 shows the execution time of the explicit Laplace equation solver. The results for one node (an ordinary multithreaded version) show very low scalability for the number of threads, because the performance of the memory bus becomes bottlenecked because the data is so large. As the number of nodes increases, the amount of local computation decreases, but the message size of internode communication does not depend on the number of nodes. The speedup on eight nodes exceeds the linear speedup, because the working set becomes small enough to fit in the cache and the NICAM provides fast internode communication (see Figure 5a).

CG KERNEL

Figure 5b shows the speedup of the CG kernel (class A, the problem size defined in the NAS parallel benchmark) on COMPaS. 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 the CG kernel is large and main-memory accesses are frequent, the performance of the memory bus becomes a bottleneck.

While the main kernel loop of a matrix-vector multiply requires 120 Mbytes/s memory bandwidth to fully utilize the Pentium Pro processor, the total memory bus bandwidth for a read from each processor in the same node is 208 Mbytes/s shown (see Table 1). This is why the performance scales only up to two threads in each node. The same memory bus bottleneck was found in memory-intensive applications, such as in a radix sort algorithm.
**Matrix-Matrix Multiplication**

Figure 5c shows the speedup of the matrix-matrix multiplication. Although the matrix is too large ($1,800 \times 1,800$) to fit into the cache, our blocking and tiling algorithm, which uses the cache effectively, provides high performance on one node. The efficiency for the eight-node and four-thread case reaches approximately 80%.

Fast communication is also important for scalability. For example, in a matrix multiply on eight nodes, the local computation (submatrix multiplication by threads) takes 17.09 seconds, 8.59 seconds, and 4.46 seconds for one, two, and four threads. Although the local computation is scalable on an SMP node, the time for internode communications is 0.83 seconds, which does not depend on the number of threads. As the number of threads increase, the internode communication time is revealed more clearly—even if we use the fast communication layer, NICAM.

Both exploiting high cache locality to reduce memory bus traffic and high-performance internode communications enable high performance with hybrid programming. These are also effective in parallel LU factorization, which can adopt the blocking algorithm in matrix multiplication to remove a memory bandwidth bottleneck.

**Block Sparse Cholesky Factorization**

This kernel (taken from Splash-2 shared-memory programs) is a typical irregular application with a high communication-to-computation ratio and no global synchronization. The original program uses memory-lock operations in a shared-memory space to access the task queue exclusively. To implement the sparse Cholesky factorization for SMP clusters, an asynchronous message posts the message to notify that some task is ready in a different node. The one-sided communication reduces the synchronization overhead on locks and allows the communication to overlap with the computation of threads. The nonzero matrix elements can be shared among the threads running in the same node. Figure 6 shows the performance on different numbers of nodes and threads. When the number of processors increases, the load imbalance degrades the performance.

To tolerate internode communication latency, we are investigating a parallel-programming scheme that overlaps communication and computation, using NICAM. Our results show the internode communication time is hidden when many threads are running. In such cases, the overlapping technique should provide high performance.

---

**Figure 4.** The execution time of the Laplace equation solver (640 matrix size). The 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.

---

**Figure 5.** Speedup of the (a) Laplace equation solver (640 matrix size); (b) conjugate gradient kernel (class A); and (c) matrix-matrix multiplication (1,800 matrix size).
So far, we have used the hybrid shared-memory/distributed-memory programming to exploit the potential performance of SMP clusters. In general, however, an SMP cluster often complicates parallel programming because the programmer must take care of both programming models. A high-level programming language, such as OpenMP and HPF, should hide this complication. We are developing an OpenMP compiler of Fortran, C, and C++ to provide a comfortable, high-performance programming environment for the SMP cluster.

Although OpenMP (http://www.openmp.org/) was originally proposed as a programming model for shared-memory multiprocessors, we extend the OpenMP model for an SMP cluster by a "compiler assisted" distributed-shared-memory system. Our OpenMP compiler instruments an OpenMP program by inserting remote communication to maintain memory consistency among different nodes and provides a view of a shared-memory model for the SMP cluster. Different from multithreaded programs on conventional software DSMs, the OpenMP program is so well-structured for parallel programming that it lets the compiler analyze the static extent of a parallel region for the optimization of efficient remote memory access and synchronization.

Although an SMP cluster offers compact packaging and networking and good maintainability, the Pentium Pro PC-based SMP node might suffer from severely limited memory bandwidth. This can result in poor performance—especially in memory-intensive applications. We are now building the next SMP cluster, COMPaS II, with the new Intel Xeon processor. Because Xeon’s chip set provides greater memory bandwidth, we expect that the new cluster will decrease the limitations of memory bandwidth for high-performance computing.

**References**


**Acknowledgments**

I thank Yoshio Tanaka, Motohiko Matsuda, Kazuhiro Kusano, and Shigehisa Sato in the Parallel and Distributed System Performance Laboratory, as well as the COMPaS group members and colleagues in the RWCP Tsukuba Research Center for their fruitful discussions. The Parallel and Distributed System Software Laboratory, led by Yutaka Ishikawa, constructed the RWCP PC Clusters I and II. Special thanks to Junichi Shimada, the director of RWCP Tsukuba Research Center, for his support.

**Mitsuhisa Sato** is chief of the Parallel and Distributed System Performance Laboratory in the Real World Computing Partnership, Japan. His research interests include computer architecture, compilers, and performance evaluation for parallel computer systems, and global computing. He received his MS and PhD in information science from the University of Tokyo. He is a member of the Information Processing Society of Japan (IPSJ) and the Japan Society for Industrial and Applied Mathematics (JSIAM). Contact him at RWCP Tsukuba Research Center, Tsukuba Mitsui-Blg. 16F, 1-6-1 Takezono, Tsukuba, Ibaraki 305-0032, Japan, msato@trc.rwcp.or.jp.