Performance evaluation of SNAIL: a multiprocessor based on the Simple Serial Synchronized multistage interconnection network architecture

Junji Yamamoto, Takashi Fujiwara, Takuji Komeda, Takayuki Kamei, Toshihiro Hanawa and Hideharu Amano

Keio University, 3-14-1, Hiyoshi Kohoku-ku Yokohama 223-8522 JAPAN

Abstract

Simple Serial Synchronized (SSS) Multistage Interconnection Network (MIN) is a novel MIN architecture for connecting processors and memory modules in multiprocessors. Synchronized bit-serial communication simplifies the structure/control, and also solves the pin-limitation problem.

Here, design, implementation, and evaluation of a multiprocessor prototype called the SNAIL with the SSS-MIN are presented. The heart of SNAIL is a prototype 1 μm CMOS SSS-MIN gate array chip which exchanges packets from 16 inputs at a 50MHz clock speed. The message combining is implemented with only a 20% increase in hardware.

From empirical evaluation with some application programs, it appears that the latency and synchronization overhead of the SSS-MIN are tolerable, and the bandwidth of the SSS-MIN is sufficient.

Keywords: Multistage Interconnection Networks, Message Combining, Multiprocessors, VLSI Implementation, Empirical Evaluations

1 Introduction

Multistage Interconnection Networks(MINs) have been well researched as an interconnection mechanism between processors and memory modules of a multiprocessor, particularly for middle scale multiprocessor. Most of them are blocking networks like the omega network, and packets are transferred in the store-and-forward manner between switching elements through bit-parallel(8-64bits) lines. In blocking networks, packets may collide each other even if their destinations are different. In this case, one of the conflicting packets is stored in the packet buffer equipped with each switching element.

However, in the conventional MINs, their complicated structure and pin-limitation problem have been a stumbling block to implement. To support the store-and-forward operation, buffer storage and the control mechanism are required in each switching element. Because bit-parallel communication requires large number of pins, large area is required to implement the large size system. Since that, the clock frequency is limited (10–50MHz).

As an alternative approach to solve these problems, we proposed a novel Multistage Interconnection Network (MIN) architecture called *Simple Serial Synchronized* (SSS)-MIN in 1992. In the SSS-MIN, all packets are transferred in the serial and synchronized manner. Synchronized bit-serial communication simplifies the structure/control, and also solves the pin-limitation problem. With the simple structure, a highly integrated chip which works with a high frequency clock rate can be utilized. In order to enhance the performance, the mechanisms of *pipelined circuit switching* and *stage hopping* are introduced.

Moreover, message combining can be implemented with a small increase in hardware using a technique called *bit-serial combining.* As a result, the SSS-MIN achieves comparable or superior performance to the conventional ones with a much smaller hardware cost.

Throughput, latency and congestion in the SSS-MIN have been analyzed with a theoretical model and computer simulation[8], and advantages over a conventional MIN are demonstrated elsewhere[2]. However, these results only show the approximate estimation because they are not based on actual parallel applications. In order to evaluate and demonstrate the performance of the SSS-MIN, we first developed a prototype LSI chip for the SSS-MIN in 1993. Using these chips, a prototype multiprocessor called SNAIL (SSS Network Architecture ImpLementaion) was developed in 1994[15]. SNAIL consists of 16 microprocessors and 16 memory modules which are connected with 4 SSS-omega chips. Several parallel applications from SPLASH/SPLASH-2, and a simple operating system called EULASH[20] have been developed on it.

In this paper, a design, implementation, and empirical evaluation of SNAIL are presented[1]. Although the technology used in SNAIL is outdated, the empirical evaluation gives us useful observation for designing the next generation machine. In Section 2, the concept, structure and control of the SSS-MIN are introduced. In Section 3, the design and implementation of the prototype LSI chip and SNAIL are described. In Section 4, performance of SNAIL is evaluated when parallel applications are directly implemented without an operating system. A simple operating system EULASH is introduced and performance with EULASH is evaluated in Section 5.

[1] The initial evaluation results were presented in [16]

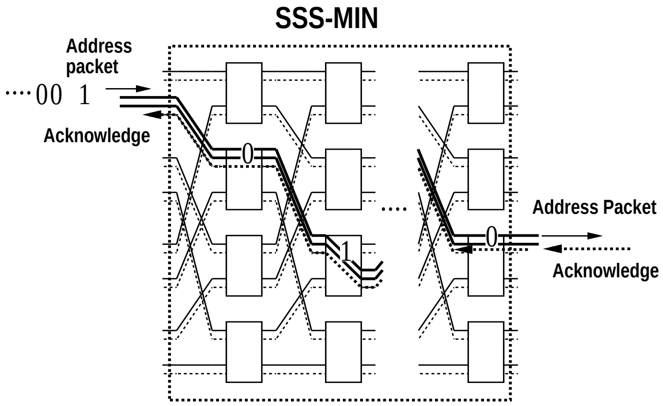


Fig. 1. Structure of the SSS-MIN

2 The SSS-MIN

2.1 Basic operation of the SSS-MIN

The basic operation of the SSS-MIN is illustrated in Figure 1. Like the Asynchronous Transfer Mode (ATM)-based packet switching systems for telecommunication, all packets are inserted into the SSS-MIN serially synchronized with a common frame clock. Since each switching element stores only one bit (or a few bits) of the packet, the SSS-MIN behaves like a set of shift registers with a switching capability. After a delay for passing through all stages, the packet headers come out at the output of the MIN.

When a conflict occurs, one of the conflicting packets is routed in an incorrect direction since the SSS-MIN provides no packet buffers in each switching element. The conflict bit in the routing tag is set when the conflicting packet is routed in the incorrect direction. Such a packet is treated as a *dead packet*, and never interferes with other packets.

When the first flit of an address packet reaches the output of the MIN, all switching elements are set to be either straight or exchange, and a path is formed from the input to the output of the MIN. Here, this path is called the *trace*.

When the *trace* is set, it can be judged whether the packet is correctly routed or not by referring the conflict bit at the output of the MIN. An acknowledge signal (ACK: ACKnowledge or NAK: Not AcKnowledge) is returned to the processor which issues the access request through this *trace* without being stored at each switching element, experiencing only the delay of wires and multiplexors. If the NAK signal is returned, the packet is re-inserted from the input buffer again at the next frame .

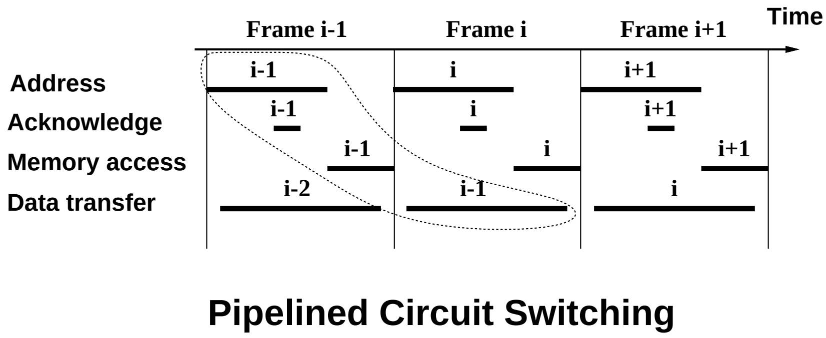


Fig. 2. Timing diagram of packets/signal transfers in the SSS-MIN

2.2 Pipelined circuit switching

The *trace* is utilized not only for the acknowledge signal but also for the transfer of data packets. The address transfer, acknowledge signal and data transfer can be performed in an overlapped manner. Figure 2 shows a timing diagram of these transfers. In frame i, an address packet is transferred. When the head of the address packet arrives at the output of MIN, the *trace* is established and the acknowledge signal is transferred immediately. If the ACK signal is returned, the data packet is also transferred using the same *trace* in frame $i + 1$. If the pass through time of the MIN is large, a pipeline can be formed for these transfers. We call this operation overlapped or pipelined circuit switching.

In practice, the depth of the pipeline is usually two since the pass through time of a packet is reduced by the stage hopping mechanism described later.

By sharing the *trace*, the structure of the SSS-MIN becomes simple unlike the conventional method which requires two independent MINs for bi-directional transfer. Figure 3 shows the structure of each switching element.

In order to allow the overlapped/pipelined operation, an address packet, a data packet, and the ACK/NAK signal use their own signal lines and multiplexors in a switching element. In the SSS-MIN, the state of an element (straight or exchange) is set by the address packet header. Since the ACK/NAK signals and data packet share the same *trace*, which is set by the address packet, only one controller is required on the address path.

2.3 Bit serial message combining

Message combining[7] is a technique which merges similar references into composite combined references at each stage of the network. Although a cost effective switch architecture with combining has been researched and imple-

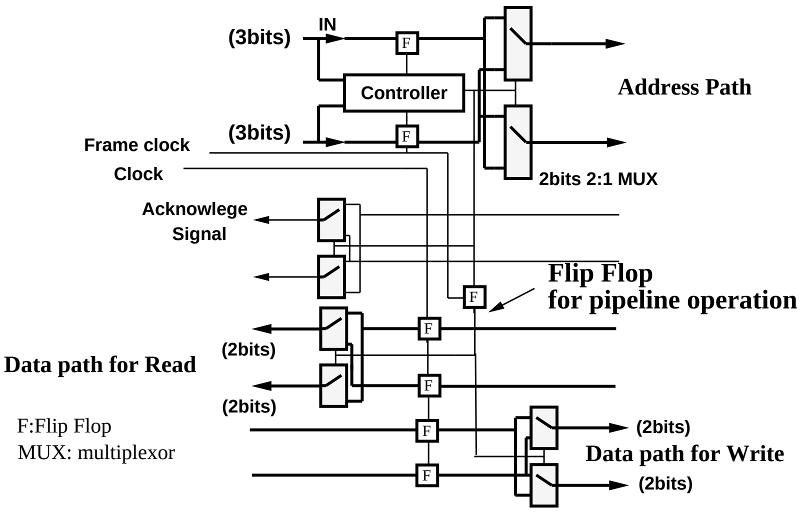


Fig. 3. Structure of a switching element in the SSS-MIN

mented[5] for conventional MINs, it still requires a large amount of additional hardware.

In the SSS-MIN, message combining is easily performed by a bit-by-bit comparison during the address packet transfer. If the addresses of two packets are the same, the switching element is set to be the backward broadcast mode B as shown in Figure 4a. In this example, three read references for the memory module 1 are combined together. After these address packets have passed by, the tree-like *trace* remains in the switching elements, and the returning data packets are multicast through the tree. This technique is also useful for write-read combining.

To combine Test&Set operations, only a small amount of additional hardware is required. The tree-like *trace* for data transfer is also formed as shown in Figure 4a. However, the data is not multicast but one of the returned values is forced to '1' in each element as shown in Figure 4b. At the same time, the Memory Interface Unit (MIU) which is provided between the MIN and the memory modules writes all-1 into the target address of the memory module. As a result, only one processor can read out the data in the target address, and other processors get all-1. This mechanism is called bit-serial message combining[1].

2.4 Stage hopping

Once the *trace* is established, the rest of the address packet and the data packet do not need to be stored in each switching element. Thus, the flip-flop can be omitted in some stages, and packets can hop over several stages. The latency is much reduced by this mechanism, called *stage hopping*.

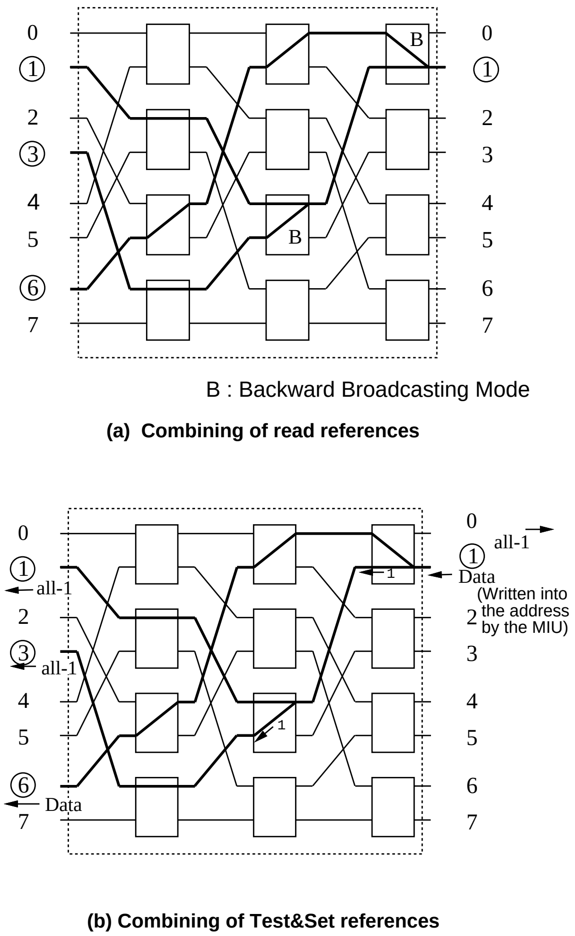


Fig. 4. Bit serial message combining

2.5 Tandem Banyan Switching Fabrics

Since one of the conflicting packets must be inserted again in the next frame, packet conflicts inside the MIN will severely degrade performance of the SSS-MIN. In order to reduce this, a network with large bandwidth is required for the SSS-MIN.

Non-blocking networks like the crossbar, Clos[4], Batcher-Banyan[13] and BDOC[6] can not support enough bandwidth because packets for the same destination collide with each other inside the network. This type of conflict limits the pass-through ratio (bandwidth) of packets for random destinations (about 65%). In order to obtain a large pass-through ratio, the MIN must provide multiple outputs for passing multiple packets for the same destination.

Tandem Banyan Switching Fabrics (TBSF)[14,19] proposed for the ATM-based packet switching system is a network with such a facility.

As shown in Figure 5, it consists of placing a number of Banyan interconnection

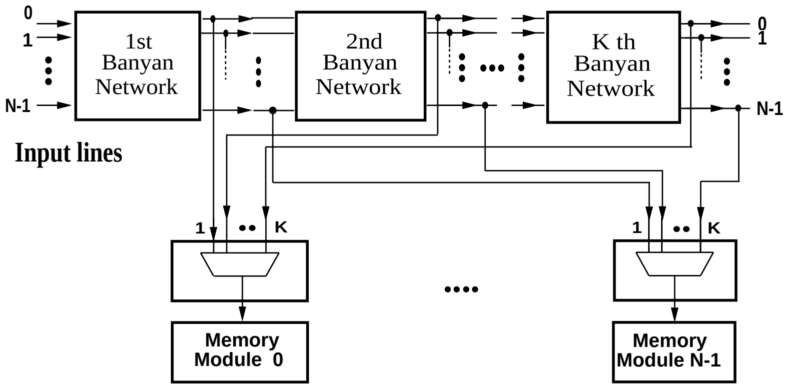


Fig. 5. Tandem Banyan Switching Fabrics

networks in series such that, for each output of each Banyan network, there is a connection feeding to the corresponding memory module, and a connection feeding the corresponding input of the following Banyan network. At the end of a Banyan network, all those packets which have succeeded in reaching their desired destination proceed to the memory modules; all the misrouted packets, after resetting their conflict bit, are fed to the next Banyan network. The interface for the memory modules provides small packet buffers for each output of the Banyan network. When the buffer is full, the NAK signal is returned even if the packet is correctly routed to the memory module.

Compared with other non-blocking networks, the TBSF has the following advantages:
Throughput: Although the network topology is not completely non-blocking, the throughput is superior to those of other non-blocking networks [19] because multiple packets for the same destination can be delivered.

Latency and Hardware: The latency of the TBSF is much smaller than those of other nonblocking networks such as Batcher-Banyan[13], the BDOC[6], and it requires a smaller amount of hardware than those of the crossbar and Clos network[3].

Memory utilization: One of problems of the TBSF used in the ATM packet switching is that the difference in the arriving time between packets depends on the number of Banyan networks which the packet passed through. However, when it is used in the SSS-MIN, this problem changes into an advantage. A memory module can be accessed as soon as a packet arrives at the module, and the memory utilization ratio is improved.

While the throughput of the TBSF is improved as the number of Banyan networks increases, the latency or the frame period is stretched. Therefore, the number of Banyan connected in tandem must be decided taking with system size, memory access time, processor performance, and other issues on the multiprocessor in which the SSS-MIN is used into consideration. Results

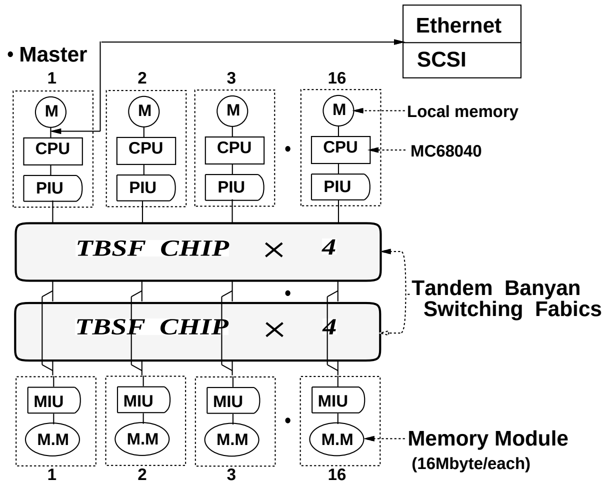


Fig. 6. Diagram of a multiprocessor SNAIL

of the computer simulation show that the optimal number of Banyans is two or three in most multiprocessors with 256 processing units[2]. The simulation results also show that the performance of the SSS-MIN with optimal TBSF is comparable to that of the conventional MIN with almost 1/8 of the chip number or 1/8 the cost[2].

3 Multiprocessor prototype: SNAIL

3.1 Outline of SNAIL

In order to evaluate the performance of the SSS-MIN with practical application programs, a multiprocessor prototype called SNAIL (SSS Network Architecture ImpLementation) was developed. As shown in Figure 6, 16 processors with local memory are connected with the 16 shared memory modules using the SSS-MIN. A processor is connected with a host workstation through the Ethernet. A performance monitor/debugger system which works independently from SNAIL is connected to both sides of the MIN. All system modules share a single system clock (12MHz) and a frame clock (12/16=750KHz) for synchronous operation.

SNAIL is the first machine providing message combining with more than 10 processors. The main purpose of SNAIL is to evaluate the performance of the SSS-MIN.

SNAIL provides the following functions:

– All processors are synchronized with the same clock used in the SSS-MIN.

The DRAM refresh operations of every shared memory module and the I/O operations are also synchronized with a frame clock of the SSS-MIN. Since the behavior of packets in the SSS-MIN can be predicted accurately, there are no uncertain factors in the system. In such a system, most synchronization code can be omitted by instruction level scheduling. Moreover, a parallel program behaves in the exactly same manner whenever it is executed.

- A prefetch operation and write buffers are provided to hide latency to access shared memory.
- Block transfer operations between the on-chip cache and shared memory make time to access shared memory short.
- In multiprocessors using conventional MINs, it is almost impossible to analyze traffic for the shared memory using a hardware monitor since too many lines must be monitored. However, in the SSS-MIN, packets are transferred through a small number of wires, and it is possible to analyze the traffic by monitoring them. Using this monitoring mechanism, breakpoints on the shared memory address can be provided.

  When a processor accesses a certain shared memory address, the network is frozen, and processors are interrupted. These functions will help performance analysis and debugging of the scheduled code.

Flexible implementation is selected in SNAIL rather than high performance. The PU-MIN interfaces and memory controller are implemented with Xilinx FPGA XC3090s. Although the operation speed of the the FPGAs is not high enough, the hardware configuration can be changed just by rewriting the configuration data.

Before looking at a detailed description of the SSS-MIN used in SNAIL, processing units and memory modules are briefly described below:

- Processing Unit: Motorola's MC68040 is used. The clock used in the SSS-MIN (12MHz) is also used as the system clock of all processors. In the current implementation, each processor is provided with 1Mbyte SRAM as local memory.

  Two FPGA chips (Xilinx's XC3090) are used for the PIU (Processor-MIN Interface Unit) which manages parallel/serial conversion between processor buses and the SSS-TBSF chips. A 2-word address packet buffer and a 2-word prefetch buffer are provided inside the PIU. If the NAK signal is returned from the MIN, the address packet is automatically inserted again from the address packet buffer.

  A processor is connected with the host workstation through the Ethernet, and behaves as a master processor. The master processor also provides the SCSI interface for disks, and the ROM for the bootstrap.
- Memory modules: Sixteen 4Mbyte-DRAM memory modules form a 64Mbyte-shared memory system which is interleaved in the line size of the on-chip cache (4 words).

Table 1
The specification of the SSS-TBSF chip

Clock: 50MHz	Bit width: 3bit(address)2bit(data)
Size: 16-input/output	Maximum bandwidth: $250 Mbits/sec \times 16$
Number of utilized cells: 9838	Cell utilization ratio: 49%
Number of signal pins: 180	Technology: $1.0 \mu m$ CMOS sea-of-gates

The MIU manages parallel/serial conversion and other control functions of memory modules including synchronization operations, block transfer, and memory refresh. It also manages two synchronization operations, Test&Set and Fetch&Dec for every memory word. In order to eliminate uncertain factors of the system, all memory modules are refreshed at the same time, synchronized with the frame signal. The MIU also consists of a Xilinx FPGA XC3090.

3.2 The prototype SSS-TBSF chip

An SSS-TBSF gate array chip which provides all functions including bit-serial message combining is used in SNAIL. [2]. An omega [11] network with 16-input/16-output is implemented into a single chip, and the TBSF can be easily formed by tandem connection. An omega network with 256-input/256-output also can be constructed with a 16×2 connection, and so 64 chips are required to form the TBSF with two Banyan networks.

The specifications of this chip are shown in the Table 1.

As shown in Figure 7, the chip consists of the input interface, the network itself, and the output interface. The input interface provides a small amount of storage to store the header of address packets during the setting of the trace. The conflict bit of valid packets is also reset in the input interface. The network is a four stage omega network with 32 2×2-switching elements. The output interface checks whether the packet reaches the correct output, and generates ACK/NAK signals.

The total number of utilized cells is 9838 and message combining requires only 20% of the total cells. This demonstrates that the bit-serial message combining is cost effectively implemented. A switching element comprises only 192 cells, and more than 70% of cells are used in the controller attached to the address lines.

[2] This chip was designed and fabricated with the cooperation of Kawasaki Steel Co. Ltd. Japan.

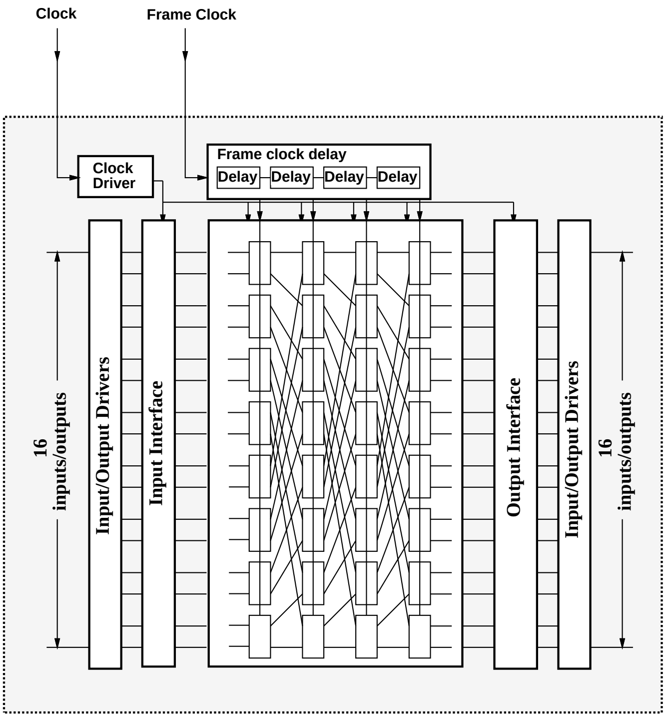


Fig. 7. Diagram of the SSS-TBSF chip

3.3 Switching mechanism in SNAIL

From computer simulation, it appears that a TBSF network with a single stage is sufficient for the configuration with a 16 processors/memory system. In this configuration, the performance is limited by latency rather than overhead caused by congestion. Thus, in SNAIL, four chips are used in the bit-slice manner in order to reduce the latency. They form a simple SSS omega network with 125Mbyte/sec maximum bandwidth for a processor/memory module, and 2Gbyte/sec in total.

A processor accesses shared memory with the following operations:

– Read: The processor must wait until the data is fetched from the memory.
– Write: The processor can execute the next instruction immediately unless the address packet buffer (2 packets) is full.
– Prefetch: Like the write request, a processor can execute the next instruction. The data is transferred into the prefetch buffer (2 words) after the latency of the MIN. The processor can read out the data from the prefetch buffer without any delay if the data has reached. Otherwise, the processor must wait in the same way as the read operation.
– Block transfer: When a processor uses its on-chip data cache as the shared memory, a cache line (4 words) is transferred quickly using the page access mode of the DRAM. It is useful for array calculation and quick data transfer

Table 2
Access latency

<table>
<tr><td colspan="2">Access type</td><td>clock</td></tr>
<tr><td colspan="2">Local memory</td><td>1</td></tr>
<tr><td rowspan="3">Shared memory</td><td>write</td><td>1</td></tr>
<tr><td>read</td><td>12</td></tr>
<tr><td>sync.</td><td>12</td></tr>
</table>

between shared memory and local memory.
– Synchronization: Test&Set and Fetch&Dec operations are supported in SNAIL. In addition, SNAIL provides another simple barrier synchronization mechanism which does not require the access of shared variables through the SSS-MIN.

Bit-serial message combining is performed among read operations, prefetch operations, and Test&Set operations. Since the decrement operation of the Fetch&Dec executes in the MIU, it cannot be combined in the SSS-MIN.

3.4 Access latency

Table 2 shows the access latencies of the SNAIL system. The reason why the shared memory write can be done in 1 clock is the existence of a write buffer. The latencies of shared memory accesses shown in this table are the best case. If packet resending (due to packet conflict) or synchronization to frame (the period is 8 clocks) are performed, the latencies become longer.

3.5 Implementation

As shown in Figure 8, SNAIL consists of four CPU boards and a backplane board.

Four CPUs and PIUs are mounted on a single CPU board, and they are connected through connectors with four SSS-omega chips on the backplane board. The backplane board also includes 16 memory modules and MIUs, and its size is 70cm $\times$ 40cm.

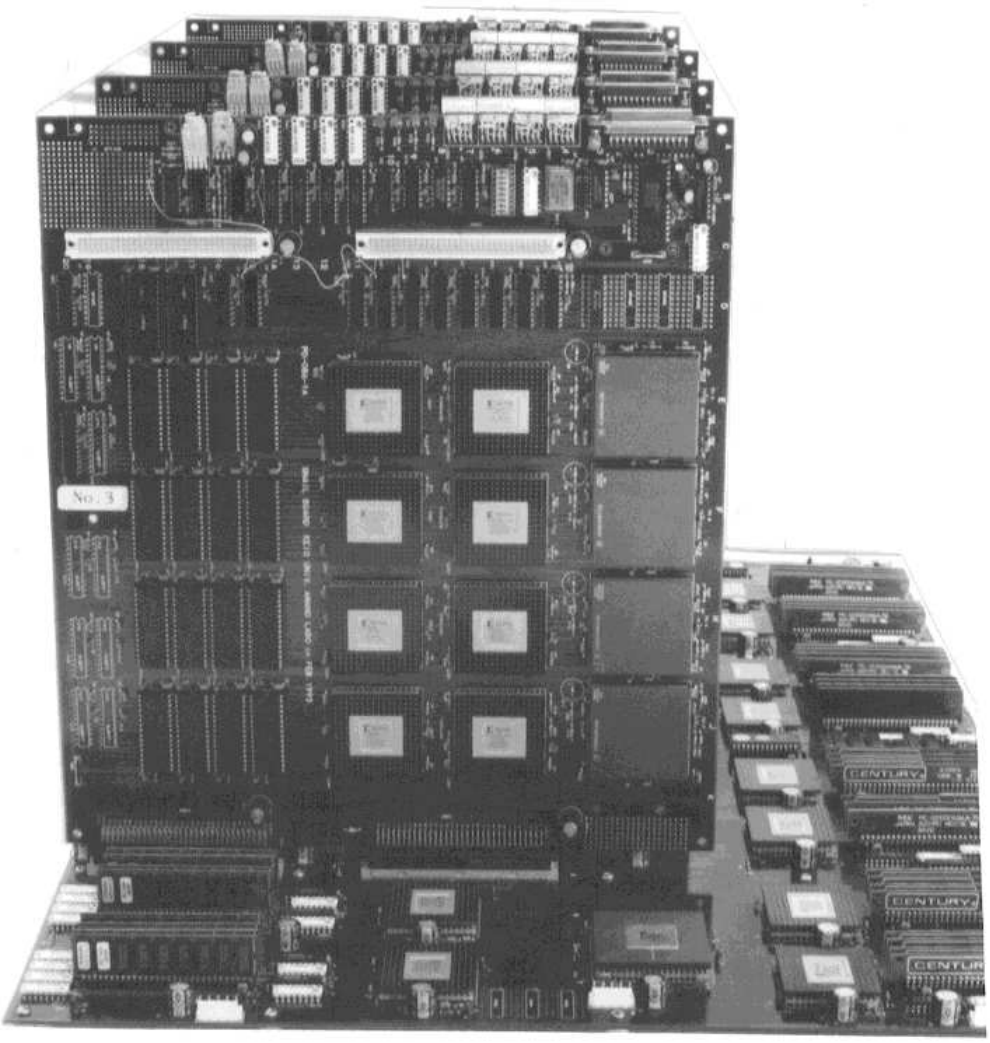


Fig. 8. SNAIL

4 Empirical evaluation of bare machine

In this section, the empirical evaluation results of SNAIL without an operating system is described. Since these results are based on measurements of real hardware, the evaluation is limited by the the prototype hardware limitations such as the poor performance of the processor and the small size of the system. However, this kind of measurement is still useful since we can observe how performance is balanced between the processor and network under the behavior of real application programs. The efficiency of the message combining and prefetch mechanism can also be evaluated under real conditions. These results are useful for the design of the next generation machine with advanced processors and SSS-MINs.

Table 3
Overview of Application Characteristics

application	contents	algorithm	scheduling	synch.†	gran.‡
Water	molecular dynamics, short-range N-body	direct, with cutoff radius	static	barrier	large
Barnes-Hut	astrophysics, hierarchical N-body	Barnes-Hut	dynamic (programmer)	barrier,flag	large
MP3D	aeronautics, particle-in-cell	Monte Carlo	static	barrier	large
Cholesky	sparse matrix factorization	supernodal fanout	dynamic (task queue)	task queue	large

†synchronization
‡granularity

4.1 Application programs

As a benchmark, we selected six applications: Water, Barnes-Hut, MP3D, Cholesky, Jacobi and Neuro shown in Table 3. The first four applications are from SPLASH (Stanford Parallel Applications for Shared-Memory) [17] suites developed at Stanford University. The rest are original applications developed on SNAIL.

Variables which are actually shared between processors are assigned in the shared memory. Instructions and local data are distributed into the local memory of each processor when the program starts. The barrier synchronization used in the program is translated into the hardware barrier operation. The Lock/Unlock operations used for management of the task queue are executed with a Fetch & Dec operation. The on-chip instruction cache is enabled in all executions.

4.2 The conditions of evaluation

4.2.1 Parameters

Water Input file LWT12 is used. The number of molecules is set to be 64 and other parameters are set as follows: temperature = 298.00K, density = 0.99800G/C.C., time step = 1.50E-16sec, order used to solve F=ma = 6, number of time steps = 2, spherical cutoff radius = 6.28128 Å.

Barnes-Hut Parameters are set as follows: infile = “”, nbody = 1024, seed

= 123, dtime = 0.025, eps = 0.05, tol = 1.0, fcells = 2.0, tstop = 0.075(4 step), and dtout = 0.25. *Costzones* is used for partitioning.

MP3D Due to limitation of the memory size, system with 500 particles in 14x24x7 space is simulated. 50 simulation steps are measured.

Cholesky Input file is a 3466-by-3466 matrix with 10215 non-zeros in the matrix. It has 1846 distinct super-nodes.

Jacobi It solves simultaneous linear equations (200×200 tridiagonal matrix) with Jacobi's iterative method.

Nuero The 16-Queen problem is solved with the Hopfield neuron model[18]. 16×16 neurons are utilized.

4.2.2 Performance indices and monitoring system

In SNAIL, the performance monitor board, which can be independently plugged into either side of the SSS-MIN (that is, processor side or memory side), is designed to count various events in the system without affecting any operation of the processors. The board provides a Motorola 16bit CPU (MC68000) for the interface and a Xilinx FPGA XC3090 for event counters.

Using this board, the following events can be monitored:

- t_n: execution time,
- OUT: the total number of packets which pass through the network,
- CF: the number of conflicting packets, and
- IN: the total number of requests which are issued by the processors.

Using these values, we can obtain performance indices: the speedup ratio, the network utilization, the conflict ratio, and the combining ratio. The **Speedup ratio**, defined as the ratio of the execution time after increasing the number of processors to one with a single processor, is represented with t_n/t_1. The **Network utilization ratio** is defined as the expected number of active packets in a frame, and is represented as follows:

$$\frac{ACT}{N \times T_F}.$$

ACT is the total number of active packets, and can be given by $(OUT - CF)$, where N is the number of processors in the system, and T_F is the total number of frames.

The **Conflict ratio** is the ratio of conflicting packets to the total number of packets which pass through the network, and is represented in CF/OUT. The **Combining ratio** is the ratio of combining packets to total packets. Since the number of combining packet (CB) can be given by $(IN + CF - OUT)$,

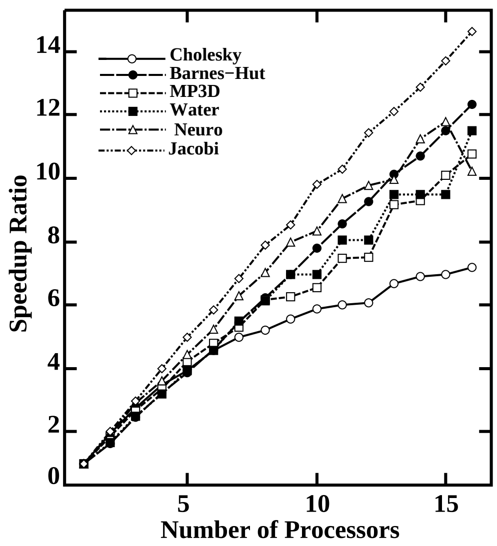


Fig. 9. Speedup ratio

the combining ratio is represented as

$$\frac{CB}{OUT} = \frac{IN + CF - OUT}{OUT}$$

4.3 Performance evaluation

4.3.1 Speedup ratio

As shown in Figure 9, almost linear speedup is achieved and it is more than 10 times faster than that with the single processor except for Cholesky. As mentioned above, shared variables are stored in main memory even when the program is executed with a single processor in this evaluation. Thus, overhead is mainly caused by the following reasons:

– the increase of the latency caused by network congestion,
– the loss caused by the synchronization operation itself, and
– the waiting time caused by the load unbalance and insufficient parallelism of the program.

Since the number of jobs which can execute in parallel is sufficiently larger than the number of processors, waiting time caused by the load unbalance is not large in this situation. Cholesky only uses the synchronization with the task queue, and this is a reason why the speed up ratio is the worst among these programs.

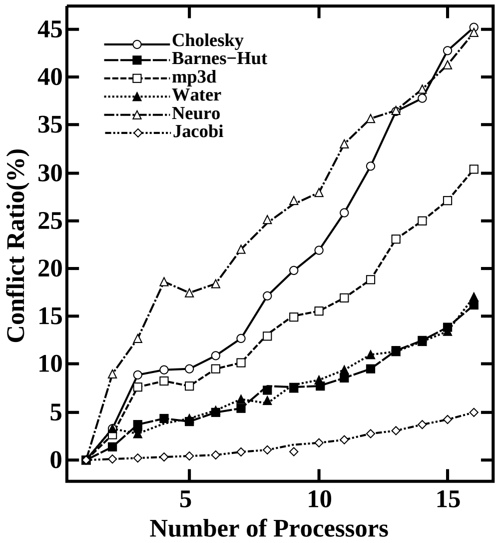


Fig. 10. The ratio of conflicting packets to all packets

4.3.2 Network congestion

Figure 10 shows the packet conflict ratio. The ratio increases almost linearly with the number of the processors. In applications which require frequent shared memory accesses (Cholesky and Neuro) this ratio reaches about 45% with 16 processors.

Figure 11 shows the network utilization ratio. In MP3D, Water and Barnes-Hut, the ratio is 25%, 12% and 8% respectively, and it is almost independent from the number of processors. In these applications, network congestion is not so severe, and performance is not degraded.

On the other hand, in Cholesky, the utilization ratio is high and increases with the number of processors. Especially when the number of processors is bigger than thirteen, network utilization is much increased. This degrades the speedup ratio in this application. One of the reason is the synchronization which is used, *task queue* only in Cholesky. The task queue is implemented with *lock* and *unlock* which uses a Fetch& Dec operation to a shared variable. Since the Fetch & Dec operation cannot be combined, the task queue causes a lot of packet conflicts which increases the the utilization ratio due to re-transmitted packets.

Although there are a lot of conflicting packets in Neuro, the network utilization ratio is not as high as Cholesky. This comes from the fact that the Neuro program behaves almost in the SIMD manner, and each processor issues packets almost at the same time in each iteration.

From these results, it appears that the performance of the SSS-TBSF chip balances with the performance of the processors in SNAIL. Although the la-

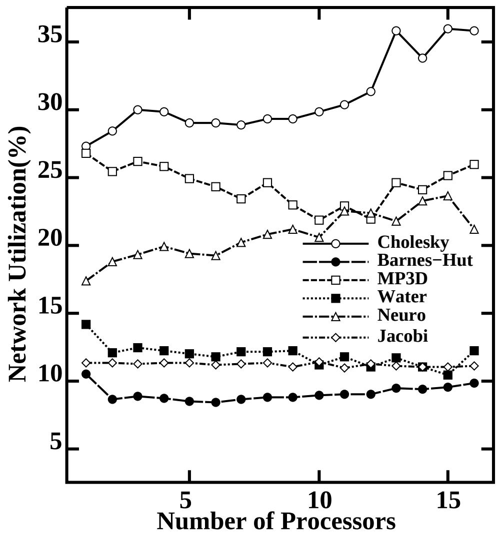


Fig. 11. The ratio of frame utilization for a processor

tency of the accessing shared variables is increased by the network congestion in some applications, the performance can still be improved by the increasing the number of processors. Thus, the network does not form a bottleneck for the system.

4.3.3 Effect of prefetch

In SNAIL, the latency of the MIN can be hidden using the prefetch operation. In **Jacobi**, each processor reads a vector element stored in the shared memory and repeatedly multiplies it with a matrix element in the local memory. Using the prefetch operation, the read request for the "i+1"th element can be issued before the calculation of the "i"th element. Figure 12 shows the performance improvement with such a prefetch operation. From 5-10% performance improvement can be achieved.

However, in order to use the prefetch operation, the program must be rewritten so as to refer to prefetch registers. Since SNAIL provides only two prefetch registers, it was difficult to rewrite other programs that use more complicated data structure. Because address translation to use prefetch registers is required, the overhead of the translation often decreases the performance. The prefetch mechanism of SNAIL should be improved so as to handle large-sized of data, such as a cache line.

4.3.4 Effect of message combining

Figure 13 shows the ratio of combined packets to all packets. The combine ratio is not so large(less than 4%) in these applications except in Cholesky, where

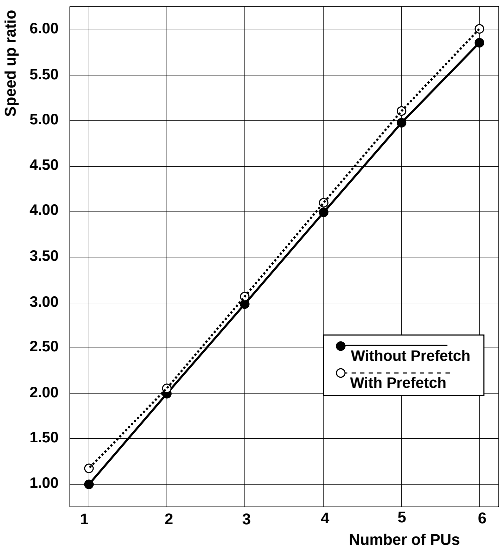


Fig. 12. Speedup ratio with the Prefetch Operation (Jacobi)

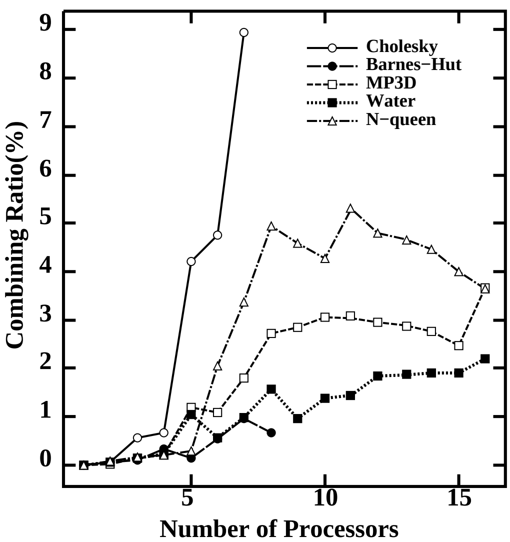


Fig. 13. The ratio of Combined packets to all packets

the combine ratio increases with the number of processors. This comes from the frequent access and high conflicting ratio in Cholesky. However, almost no improvement of execution time is observed in all applications even with 16 processors. Although a lot of packets are combined in Cholesky, most of them are for busy waiting in the task queue. Combining such packets does not improve the performance since a waiting processor cannot go to the next step until the last processor has finished its job. As a result, combining in SNAIL does not improve practical applications. This mechanism will be effective in systems with more processors.

Table 4
Execution time(sec)

		Cholesky	Barnes-Hat	MP3D	Water	Neuro
Personal Computer[1]		0.053	0.540	0.026	0.700	0.025
SNAIL[2]	GM([3])	43.90	99.40	6.10	18.30	22.000
	LM([4])	—[5]	83.90	3.52	14.12	19.50

[1]PentiumII 400MHz with RedHat Linux 5.0
[2]SNAIL with 1 processor
[3]Shared data is put in shared memory
[4]Shared data is put in local memory
[5]LM is too small to hold all the data

4.3.5 Absolute performance of SNAIL

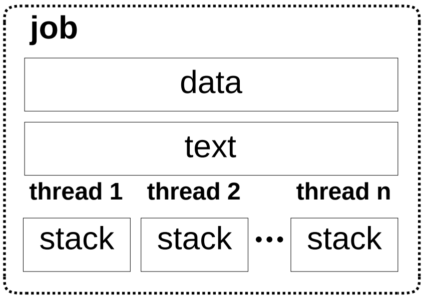
The purpose of SNAIL is to evaluate the performance of the SSS-MIN, message combining and prefetch mechanism. In the previous evaluation, the performance is shown in comparison with a single processor of SNAIL. However, even with a single processor, global variables are stored in the shared memory and the overhead of the network latency is included. In order to exclude such overhead, we located all shared variables in the local SRAM, and measured the execution time. Table 4 shows the evaluation results. As mentioned in section 3, a high speed SRAM is used in SNAIL and it can be accessed without any wait cycles. Therefore, the execution time using local memory (LM in the table) corresponds to the performance of an ordinary uniprocessor with a large secondary cache. To examine the effects of TBSF, we also execute programs whose shared data are put in local memory. As a result, the overhead using TBSF is between 13% and 20% except with MP3D which requires a large amount of global data access.

Table 4 also shows the execution time with a recent personal computer (a PC with a Pentium II). Since SNAIL was developed in 1994 and the processor used in SNAIL is three generations old, the absolute performance of SNAIL is extremely low from the current point of view.

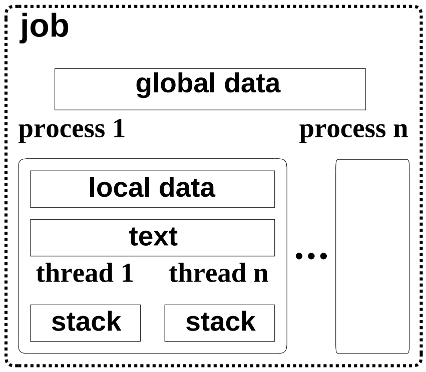
5 Performance with an operating system

5.1 A light-weight single job operating system EULASH

Although the performance without an operating system is evaluated in the above section, an operating system is required for common users, and its



(a) model for user



(b) restructured model

Fig. 14. programming models

performance is essential. Since SNAIL is a back-end machine which is used by a single user, we developed a light weight operating system for a single-user/single-job. This operating system, called EULASH[20], can make the best use of two types of memory: local and shared memory, and supports a simple but powerful programming/execution model.

In the EULASH environment, a programmer describes an application program in a manner similar to the multi-thread library of Mach or the C-thread library of the UNIX operating system. The programmer is not required to handle the data allocation nor the structure of the target multiprocessor. The EULASH environment also permits the programmer to specify detailed functions such as variable locations, thread grouping, and policy of thread context switching.

In order to minimize the loss of thread switching and reference of the shared memory with a large latency, the relationship between threads and variables is statically analyzed. Threads described by the programmer are gathered to make *processes*. In EULASH, a *process* is a virtual processor which usually corresponds to a real processor which is mapped statically. The number of processes is not equal to that of processors only when the number of processors in the target multiprocessor is changed with partitioning or system faults. As a result, the simple user programming model shown in Figure 14(a) is restructured as shown in Figure 14(b).

In order to hide the latency caused by referring variables, EULASH classifies a variable into three types: local, shared, and mobile. These types are also decided at compile time so as to make the best use of high speed local memory.

The optimized and restructured program is executed on the EULASH kernel which supports quick thread/process management and virtual memory systems. Based on the type of variables, the appropriate run-time library is

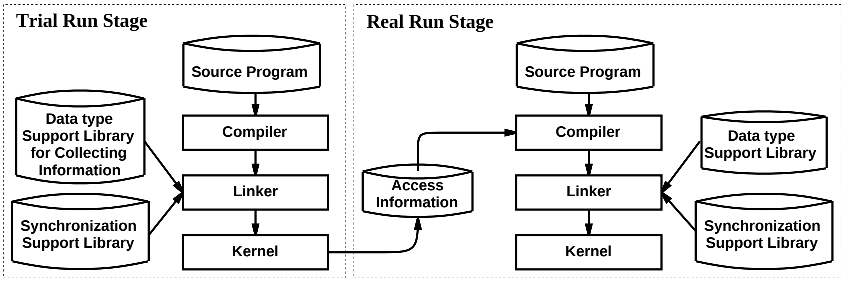


Fig. 15. General Flow of the EULASH system

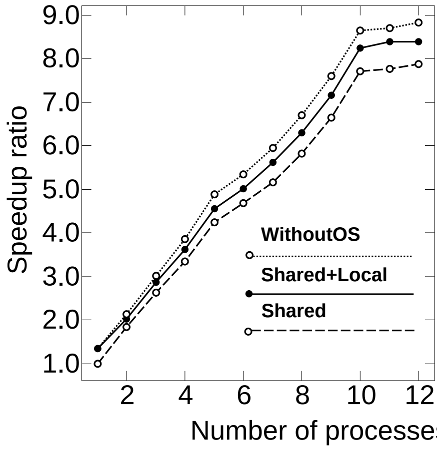


Fig. 16. Speed up ratio

called when the variables are accessed. Figure 15 shows a flow diagram of the execution of a parallel program on the EULASH.

5.2 Performance evaluation with EULASH

Figure 16 shows the speed up ratio of Jacobi which solves simultaneous equations with Jacobi's method. In this figure, "Shared" is the case where all variables are located in the shared memory as shared type, "Shared+Local" is the case where a part of the variables are converted to local variables by the EULASH compiler (In this case, mobile type is not used).

Here, "WithoutOS" shows the execution time without the EULASH kernel. Figure 16 shows that the difference between cases with and without the EULASH kernel is only 5% when the EULASH compiler optimizes the allocation of variables so as to make the best use of local memory. It also demonstrates that the overhead of the EULASH kernel is quite small.

6 Conclusion

The design, implementation and evaluation results of a prototype multiprocessor SNAIL were presented.

Through evaluations, it appears that the performance of the SSS-TBSF chip balances with the performance of processors in SNAIL. Although the latency of the accessing shared variables is increased by the network congestion with 16 processors, the network does not form a bottleneck of the system in any real applications. Even with a simple operating system, the performance degradation is quite small if the compiler allocates variables in the local memory as possible. On the other hand, the message combining mechanism and prefetch mechanism in SNAIL did not contribute much the performance improvement. The prefetch mechanism in SNAIL must be modified so as to avoid the overhead of address transform for accessing prefetch buffers. In SNAIL, the message combining mechanism works a little when the number of processors is large. Therefore, it will improve the performance in a system with more processors.

Most of the results here come from the fact that the performance of the processor used in SNAIL (68040) is not very high and system size is limited (16 processors). When more recent high speed RISC processors are used, the SSS-TBSF chip used in SNAIL may cause performance degradation. To cope with such advanced situations, we have developed a next version SSS-MIN chip called the SSS-PBSF(Piled Banyan Switching Fabrics) chip[8,10]. Introducing three dimensional structure and advanced processes (0.5μ CMOS), it supports much larger bandwidth and works at 90MHz. We also developed a network chip dedicated to cache coherent control called the MINC chip[9,12]. Using these chips, we are now developing the next generation prototype called SNAIL-2.

References

[1] H. Amano and K. Gaye. A Batcher double omega network with combining. In *Proc. of Internatinal Conference on Parallel Processing*, Aug. 1991.

[2] H. Amano, L. Zhou, and K. Gaye. SSS(Simple Serial Synchronized) - MIN: a novel multi stage interconnection architecture for multiprocessors. In *Proc. of the IFIP 12th World Computer Congress*, volume I, pages 571–577, Sep. 1992.

[3] G. Broomell and J. R. Heath. Classification categories and historical development of circuit switching topologies. *ACM Computing Surveys*, 15(2), Jan. 1983.

[4] C. Clos. A study of non-blocking switching networks. *Bell System Tech. J.*, 32(2), Mar. 1953.

[5] S. R. Dickey and R. Kenner. Combining switches for the NYU ultracomputer. In *Proc. on IEEE Frontiers to Massively Parallel Processing*, pages 521—523, Nov. 1992.

[6] K. Gaye and H. Amano. A Batcher double omega network with combining. *IECIE Transaction on information & systems*, E75-D(3):307–314, 1992.

[7] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. Mcauliffe, L. Rudolf, and M. Snir. The NYU ultracomputer - designing an MIMD shared memory parallel computer. *IEEE Transaction on Computer*, c-32(2), Feb. 1983.

[8] T. Hanawa, H. Amano, and Y. Fujikawa. Multistage interconnection networks with multiple outlets. In *Proc. of the International Conference on Parallel Processing*, pages I 1–8, Sep. 1992.

[9] T. Hanawa, T. Kamei, H. Yasukawa, K. Nishimura, and H. Amano. MINC: Multistage intefconneciton network with cache control mechanism. *IEICE Transaction on Information & Systems*, E80-D(9):863–870, Sep. 1997.

[10] T. Kamei, M. Sasahara, and H. Amano. An LSI implemenation of the simple serial synchronized multistage interconnection network. In *Proc. of Asian South Pacific Design Automation Conference*, pages 671–672, Jan. 1997.

[11] D. H. Lawrie. Access and alignment of data in an array processor. *IEEE Transaction on Computer*, c-24(12), Dec. 1985.

[12] T. Midorikawa, T. Kamei, T. Hanawa, and H. Amano. The MINC chip. In *Proc. of Asian South Pacific Design Automation Conference*, pages 337–338, Feb. 1998.

[13] M. J. Narasimha. The Batcher-Banyan self-routing network: Universality and simplification. *IEEE Transaction on Communication*, 36(10):1175–1178, Oct. 1988.

[14] H. Sakamoto, T. Masaki, H. Inoue, and H. Amano. Configuration and evaluation of self routing switches. *IN88*, (8), Jan. 1988. (in Japanese).

[15] M. Sasahara and et.al. SNAIL: a multiprocessor based on the Simple Serial Synchronized Multistage Interconnection Network architecture. In *Proc. of the International Conference on Parallel Processing*, pages I 110–117, Aug. 1994.

[16] M. Sasahara, J. Terada, J. Yamato, T. Hanawa, and H. Amano. SNAIL: a multiprocessor based on the simple serial synchronized multistage interconnection network. *IPSJ*, 36(7):1640–1651, Jul. 1995. (in Japanese).

[17] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. SPLASH: Stanford parallel applications for shared-memory. In *Computer Architecture News*, volume 20, pages 5–33, Mar. 1992.

[18] Y. Takefuji. *Neural Network Parallel Computing.* Kluwer Academic Publishers, Jan. 1992.

[19] F. A. Tobagi and T. Kwok. The tandem banyan switching fabric: a simple high-performance fast packet switch. In *Proc. INFOCOM91*, Apr. 1991.

[20] J. Yamamoto, D. Hattori, J. Yamato, T. Tokuyoshi, Y. Yamaguchi, and H. Amano. A preprocessing system of the eulash: an environment for efficient use of multiprocessors with local memory. In *Proc. of IASTED/ISMM International Conference on Parallel and Distributed Computing and Systems*, pages 68–71, Oct. 1995.