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

Masashi Sasahara, Jun Terada, Luo Zhou, Kalidou Gaye,
Jun-ichi Yamato, Satoshi Ogura, Hideharu Amano
Dept. of Computer Science, Keio University
3-14-1, Hiyoshi Yokohama 223 Japan
E-mail: {sasahara, terada, zhou, gaye, yamato, ogura, hunga}@aa.cs.keio.ac.jp

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 simpli es the structure/control, and also solves the pin-limitation problem.

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

From the 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 suf cient. Although the performance improvement with the bit serial message combine is not so large (1%) when instructions are stored in the local memory, it becomes up to 400% when instructions are stored in the shared memory.

1 Introduction

Multistage Interconnection Networks (MINs)[1] have been well researched as an interconnection mechanism between processors and memory modules of parallel machines. 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 (8bits-64bits) lines. In blocking networks, packets may collide with each other in the network even if their destinations are different. In this case, one of the con icting packets is stored in the packet buffer.

However, the implementation of large scale MINs has been a cumbersome job because of their complicated structure and pin-limitation problem. Because of the store-and-forward operation, a buffer storage and control mechanism are required in each switching element. The pin-limitation problem caused by bit-parallel communication makes the high density implementation dif cult. Since a large number of chips requires a large area for implementation, the clock frequency is limited in relatively low level (10-50MHz).

As an alternative approach, we proposed a novel Multistage Interconnection Network (MIN) architecture called Simple Serial Synchronized (SSS)-MIN[2]. In the SSS-MIN, packets are serially transferred synchronized with a frame clock like the packet switching network used in the telecommunication. Synchronized bit-serial communication simpli es the structure/control, and also solves the pin-limitation problem. The simple structure allows the use of a high frequency clock and a high throughput network topology[3]. To enhance the performance, *pipelined circuit switching* and *stage hopping* are introduced.

Moreover, the message combining can be implemented with a small increase of hardware using a technique called *bit-serial message combining*. As a result, the SSS-MIN achieves comparable or superior performance to the conventional ones with much smaller hardware cost[2].

Here, a design and implementation of a multiprocessor with prototype LSI chips for the SSS-MIN are presented. Through the empirical evaluations using some parallel applications, the performance of the SSS-MIN is demonstrated. By the limitation of pages, the detail structure and operation of the SSS-MIN are omitted here. See [2] or [3].

2 Multiprocessor prototype: SNAIL

SNAIL (SSS Network Architecture ImpLementation) is a multiprocessor prototype which is developed for evaluating the performance of the SSS-MIN with practical application programs. As shown in Figure 1, 16 processors with local memory are connected with the 16 shared memory modules using the SSS-MIN. The performance monitor and debugger system which works independently from SNAIL are connected both sides of the MIN. All system modules share a single system clock (50MHz) and a frame clock (50/16=3.125MHz) for the synchronous operation.

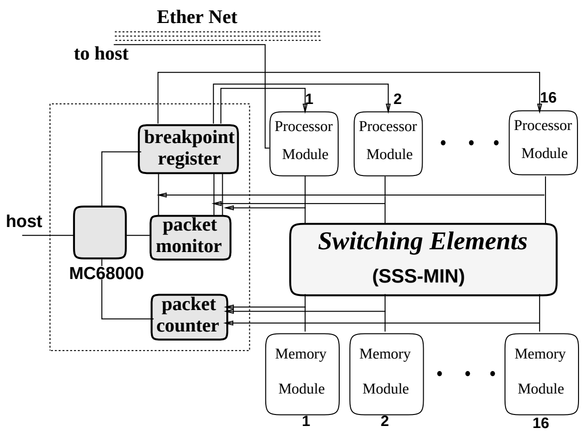

Figure 1: Diagram of a multiprocessor SNAIL

SNAIL is also designed as a test-bed of the static scheduling with the best use of the predictability and schedulability of the SSS-MIN. In the SSS-MIN, operations are simple and synchronized, and thus, highly predictable. If the parallelizing compiler and its optimizer are highly sophisticated, they can generate an instruction level scheduled code which hides the latency of the network and eliminates most synchronization operations[4].

For this purpose, SNAIL provides the following functions:

- All processors are synchronized with the same clock used in the SSS-MIN. The refresh of every shared memory module and the I/O operations are also synchronized with a frame clock of the SSS-MIN. Since the priority of con icting packets in the

SSS-MIN can be controlled, there are no uncertain factors in the system. In such a system, most of synchronization code can be omitted by the instruction level scheduling. Moreover, a parallel program behaves exactly in the same manner whenever it is executed.

- A prefetch operation, write buffers, and block transfer between shared memory and on-chip cache are supported for the shared memory. Using these functions, the optimizer can hide the latency of the shared memory access.

- In multiprocessors using conventional MINs, it is almost impossible to analyze traf c for the shared memory by 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 traf c by monitoring them. Using this monitoring mechanism, shared memory breakpoint is also prepared.

For these purposes, exible implementation is selected in SNAIL rather than high performance. The PU-MIN interfaces and memory controller are implemented with Xilinx's FPGA XC3090s.

The detail structure of SNAIL is shown as follows.

Processing Unit: Motorola's MC68040 is used. The clock used in the SSS-MIN (50MHz) is also used as a system clock of all processors. In the current implementation, each processor provides 0.5Mbyte SRAM as a local memory.

The PIU (Processor-MIN Interface Unit) manages parallel/serial conversion between processor buses and the SSS-MIN chips. 2-word address packet buffer and 2-word prefetch buffer are provided inside the PIU.

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).

The MIU (Memory Interface Unit) manages parallel/serial conversion and other controls 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.

SSS-MIN in SNAIL: The heart of SNAIL is the SSS-MIN with a prototype chip which provides all functions including the bit-serial message combining[1]. An omega [5] network with 16-input/16-output is implemented into a single chip, and a high throughput network TBSF (Tandem Banyan Switching Fabirics)[6][3] can be simply formed by the tandem connection of them. The number of cells for the message combining is only about 20% of total number of required cells.

The speci cation of this chip is shown in the following table:

This chip can be used to form a high bandwidth network TBSF with simply tandem connection of them. However, in SNAIL, four chips are used in the bit-slice manner to form a simple omega network. A processor accesses the 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.

[1]This chip was designed and fabricated with the cooperation of Kawasaki Steel co. ltd. Japan.

Table 1: The speci cation of the SSS-omega chip

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

- 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 as the read operation.

- Block transfer: When a processor uses its on-chip data cache for the shared memory, a cache line (4 words) is transferred quickly using the page access mode of the DRAM. It is useful for the array calculation and quick data transfer between shared memory and local memory.

- Synchronization: Test&Set and Fetch&Dec operations are supported in SNAIL. While Test&Set operations can be combined in the MIN, Fetch&Dec is executed only inside the MIU. SNAIL provides another simple barrier synchronization mechanism which does not require the access of the shared variables through the SSS-MIN.

The bit-serial message combining is performed between read operations, prefetch operations, and Test&Set operations.

3 Performance evaluation

Now, a prototype system with 12 processors and 4 memory modules is operational. The switching system is full size (for 16 processors/16 memory modules) of the simple SSS-omega network. Three PCB boards consisting 4 processors are connected with another PCB board which holds the SSS-TBSF chips and memory modules. Here, the initial evaluation results using the following application programs are shown:

- **Four coloring** solves the four coloring problem for 48 states of U.S.A by using the Hop eld neuron model[7]. 48×4 neurons are destibuted into processors.

- **N-Queen** solves 16-Queen problem with the similar neuron model[7]. In this case, 16×16 neurons are utilized.

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

These parallel applications are optimized for multiprocessors with local memory, thus, instructions and data accessed locally are assigned in the local memory. In order to maximize the traf c load of the MIN, all synchronization used in these applications are the barrier synchronization using the Fetch&Dec operations to shared variables.

Table 2: Memory Access time

Frame period	320nsec
32bit data read/synchronization	640nsec
1 line block transfer	880nsec

3.1 Latency of the SSS-MIN

The access time of the shared memory when a packet does not collide with other packets in the MIN is shown in Table 2. Since the local memory can be accessed with 80nsec delay, the latency of the shared memory is 8 times larger than that of the local memory.

Figure 2 shows the speed up ratio of the parallel programs normalized by the execution time with a processor. In order to show the loss of the parallel execution, results of serial versions are shown together. In the serial version, all variables and instructions are stored in the local memory, and codes for synchronization are eliminated. From

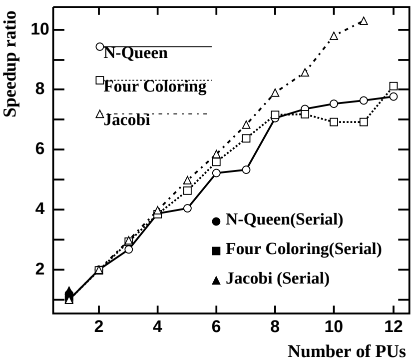

Figure 2: Speedup ratio of applications

this gure, the serial version is about 10% - 30% faster than that of the parallel version with a processor. Since the overhead of the synchronization is not large when only a processor is used, the overhead is mostly caused by the latency of the SSS-MIN. Considering that the access time of the shared memory is about 8 times of that of local memory, this overhead is reasonable.

In **Jacobi** solving a large matrix, the performance is almost linerly increases with the number of processors. In **N-Queen** and **Four Coloring**, only a small performance improvement is observed if the number of processors is more than 8. In these applications, all values of neurons must be accessed for checking convergence in each iteration. Since this convergence checking cannot be done in parallel, the performance improvement is reduced if the number of neuron assigned to each processor is small.

These results suggest that the latency of the shared memory access dose not degrade the performance of SNAIL severely. Furthermore, the latency can be hidden using the prefetch operation. In **Jacobi**, each processor reads a vector element stored in the shared memory and multiply it with a matrix element in the local memory repeatedly. 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 3 shows the performance improvement with such a prefetch operation. From 5-10% performance improvement can be achieved.

3.2 Congestion of the SSS-MIN

Figure 3.2 shows the ratio of the number of the con icting packets to all packets issued from all processors.

The ratio increases almost linearly with the number of the processors (about 1% - 3% per processor). In **Jacobi**, since the accesses to the shared memory is well interleaved, the con ict ratio is not so

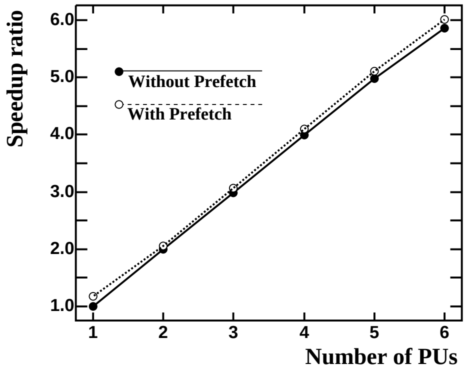

Figure 3: Speedup ratio with the Prefetch Operation (Jacobi)

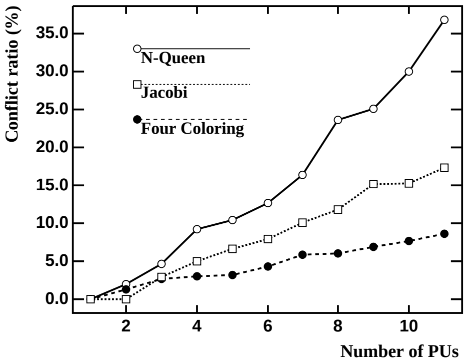

Figure 4: The ratio of the con icting packets to all packets

large. In **N-Queen** which requires most frequent access to the shared memory, it increases beyond 35%. It means that the access latency shown in Table 2 is stretched in a frame period with this probability. However, as shown in Figure 2, this stretch of the latency does not degrade the performance severely. Here, only 4 memory modules are used, and this value will be reduced when the accesses are distributed to 16 memory modules.

Figure 5 shows the average frame utilization ratio for a processor represented as the following equation.

$$\frac{A\ number\ of\ frames\ which\ a\ processor\ tries\ to\ send\ a\ packet}{A\ total\ number\ of\ frames\ in\ the\ execution}$$

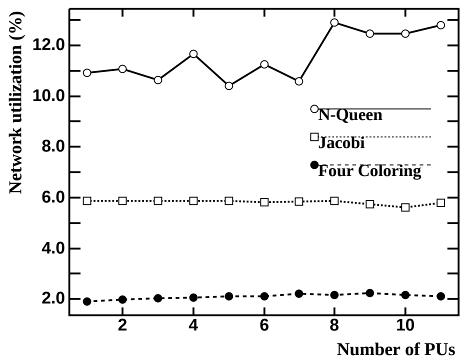

Figure 5: The ratio of the frame utilization for a processor

Although the ratio is almost dependent of the access frequency of the parallel applications, it increases by re-transmission of the con icting packets.

As shown in this gure, the utilization ratio is about 2% in Four Coloring, 6% in Jacobi and 13% in N-Queen.

In N-Queen, the utilization ratio increases about 0.5% by increasing of a processor. It is caused by increasing of the con icting packets shown in Figure 3.2. Fortunately, the increasing ratio is not so large, and in the other two applications, the utilization ratio is almost constant. It demonstrates that the bandwidth of the SSS-MIN is suf cient for these applications even with the structure consisting of a single omega network and four memory modules.

3.3 Effect of the message combining

The bit-serial message combining does not improve the execution time in all applications even with 12 processors. Considering the ratio of the con icting packets, the result is reasonable because the packets collision for the hot spot degrades little performance in these applications. Figure 6 shows the ratio of combined packets to all packets. The ratio is not so large (up to 10% with 11 processors), but increases linearly with the number of processors.

In order to increase the access frequency to the shared memory, the size of the matrix solved in **Jacobi** is decreased to be equal to the number of processors. That is, 12×12 matrix is solved when 12 processors are used. It increases the frequency of the synchronization and may cause the hot spot contention. In this case, the performance improvement is not so large, only 1%. In **Jacobi**, the computation time of a processor is almost the same, and these are almost synchronized with each other. Thus, a barrier operation succeeds with a single trial in this application. This is the reason why the message combining does not improve the performance much.

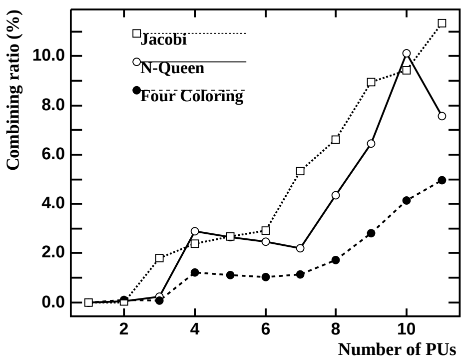

Figure 6: The ratio of Combined packets to all packets

For further increases of the access to the shared memory, we introduce a Boolean matrix calculation **BMATMUL** which requires most frequent access to the shared memory. In this application, a processor calculates a Boolean multiplication between a row of a matrix in the local memory and vector in the shared memory. Since it uses only integer operation, the access frequency is much more than that of **Jacobi** which also requires a matrix-vector multiplication but uses oating operations. Furthermore, the program is optimized considering the machine code. Registers are used as possible, and autoincrement mode of MC68040 is effectively used.

Figure 7 shows the speed up ratio with the message combining to that without the combining. Using well optimized code, the performance improvement by message combining is 1.5 - 2.0 times with more than 4 processors. On the other hand, using unoptimized code, the performance is improved only more than 8 processors, and the ratio is not so large.

These results show that the message combining does not improve the performance with the current machine scale and applications. However, it is con rmed that the number of the combining packets increases as the number of processors, and so the message combining will be effective in a larger number of processors. Furthermore, even in the current machine scale, it contributes the performance improvement if the application does not require oating operations and well optimized.

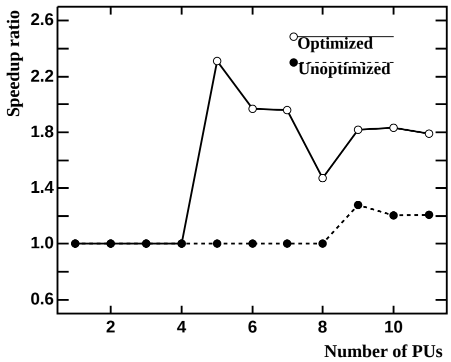

Figure 7: Speedup ratio with Message Combining in BMATMUL

4 Conclusion

From the empirical evaluation of SNAIL, it appears that the latency and synchronization overhead of the SSS-MIN are tolerable, and the bandwidth of the SSS-MIN is suf cient. Although the performance improvement with the bit serial message combine in application programs using oating operations is not so large (1%), it appears that the number of combined packets linearly increases with a number of processors. In an application with only Boolean operations, the performance becomes almost twice by using the message combining if the program is well optimized.

Now, a processor board and the block transfer facility are under debugging. Other practical applications will be evaluated on the full-scale SNAIL.

References

[1] G.Broomell, and J.R.Heath, "Classi cation Categories and Historical Development of Circuit Switching Topologies," ACM Computing Surveys, Vol.15, No.2, Jun. 1983.

[2] H.Amano, L.Zhou, K.Gaye, "SSS(Simple Serial Synchronized) - MIN: a novel multi stage interconnection architecture for multiprocessors," Proc. of the IFIP 12th World Computer Congress, Vol.I, pp.571-577, Sept. 1992.

[3] T.Hanawa, H,Amano,Y.Fujikawa, "Multistage Interconnection Networks with multiple outlets" Proc, of the International Conference on Parallel Processing, Aug, 1994.

[4] H.Kasahara, H.Honda, S.Narita, "Parallel Processing of near ne grain tasks using static scheduling on OSCAR", Proc. IEEE SuperComputing'90, pp.856-864 1990.

[5] D.H.Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Trans. on Comput. vol. c-24, No.12, Dec. 1975.

[6] F.A.Tobagi, \Fast Packet Switch Architectures For Broadband Integrated Services Digital Networks," Proceedings of the IEEE Vol.78, No.1 Jan. 1990.

[7] Y.Takefuji, "Neural Network Prallel Computing," Kluwer Academic Publishers, Jan. 1992.