A Cache Coherency Protocol for Multiprocessor Chip

Takuya Terasawa† Satoshi Ogura‡* Keisuke Inoue‡
Hideharu Amano‡

†Department of Information Networks, Tokyo Engineering University
1404-1 Katakura, Hachioji 192 Japan
terasawa@cc.teu.ac.jp
‡Department of Computer Science, Keio University
3-14-1, Hiyoshi, Kohoku Yokohama 223 Japan
{ogura,keisuke,hunga}@aa.cs.keio.ac.jp

1 Introduction

A bus connected multiprocessor is one of the most promising types of small scale parallel machines because of its simple and economical structure. Usually, all processors share a common address space of the shared memory. In order to reduce the access latency and the bus congestion, each processor provides a private cache with a snoop mechanism[1].

The existing snoop cache protocols are optimized for the current level of technologies: the access frequency of the processor, the transfer speed of the backplane bus, access latency of the SRAM used in the cache, and the bandwidth of the DRAM used in the shared memory.

However, the future advances of the devices and implementation technologies will change the structure of bus connected multiprocessors. Some number of processors and caches will be implemented on a single package using the WSI technology. Figure 1 shows the implementation of the target multiprocessor with the WSI. We call it the multiprocessor chip. In this implementation, the speed of the bus inside the package or the chip is far faster than that of the backplane bus, and the large gap of the transfer bandwidth between inside and outside of the chip will become a substantial problem. On the other hand, synchronous DRAM or Rambus DRAM with quick block transfer functions will be used in the shared memory.

According to these considerations, a protocol with following features will be effective for snoop cache systems.

- By making the best use of the powerful shared bus inside the package, efficient parallel execution of applications which request frequent data exchange will be achieved. The write update protocol is advantageous in these programs while the write invalidation protocol is still required for storing the instruction codes, stacks,

*Now he has joined Matsushita electric industrial CO., LTD.

and local data. Therefore, the mixed protocol cache[2] which allows the selection of the protocols is essential for a bus connected multiprocessor with the WSI.

- Since some number of processors and caches can be implemented in a package, the data transfer between cache modules is quickly performed. On the other hand, the communication between the caches and the off-the-package shared memory must be minimized as the latency is much larger than that of the communication inside the package.
- In order to achieve the best use of the quick block transfer functions of the synchronous DRAM or Rambus DRAM, the access to the shared memory should be performed with block transfer as possible.

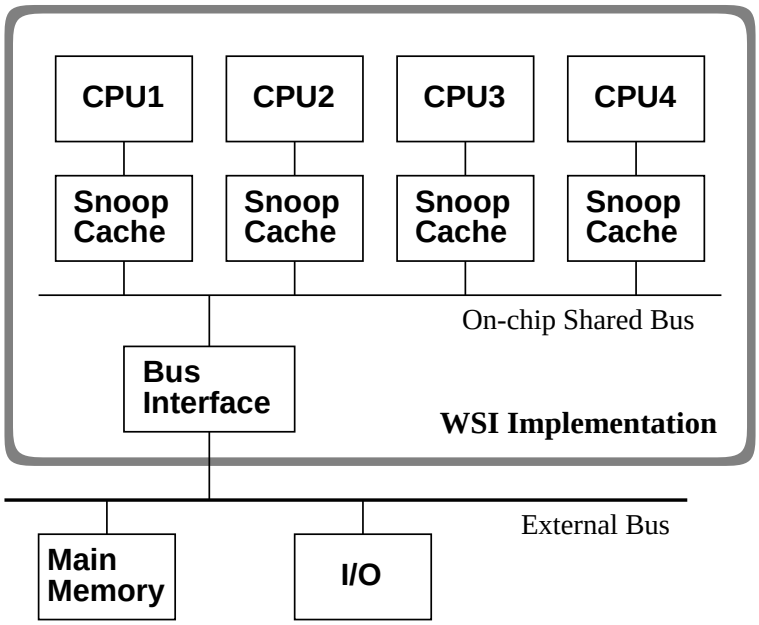

Figure 1: Multiprocessor chip

In this paper, a protocol optimized for a multiprocessor with the WSI is proposed. Then, it is extended by combining with the synchronization and message passing scheme which minimizes the shared memory access as possible. The performance of the protocol is evaluated using a multiprocessor simulator, and the efficiency is demonstrated.

2 Modified-Keio protocol - a protocol for a multiprocessor with the WSI

Most of current commercial multiprocessors use protocols like Illinois[3], which is not suitable for the WSI implementation since it requires much communication with shared memory. Therefore, we selected Keio protocol which we proposed in [4] as the basis of the protocol for a multiprocessor with the WSI. Keio protocol has the following desirable features.

- It uses cache-to-cache transfer as possible, thus, the communication between cache and shared memory is minimized.
- The communication between cache and shared memory is only required for replace and write back of a cache line, thus, the shared memory is accessed only with a block transfer mode.
- It provides a message transfer facility which can send a message between cache modules without using the shared memory.

Based on this protocol, we propose Modified-Keio protocol which supports both the invalidate and the update protocols. Modified-Keio protocol provides the following six states:

- Invalid(I)
- Clean-Exclusive-Owner(CEO)
- Clean-Shared-Owner(CSO)
- Dirty-Exclusive-Owner(DEO)
- Dirty-Shared-Owner(DSO)
- Shared-non-Owner(S)

When a missed cache block is requseted, it is supplyed by the owner of the block. It is also responsible to write back the contents of the block to the shared memory when it is replaced.

In Modified-Keio protocol, the block of the owner is exclusively selected as follows:

1. If no cache has that block, the shared memory is the owner.
2. When a block is came from the shared memory on a cache miss, the receiving cache become the new owner of the block. On the other hand, when the block is came from the other (e.g. owner) cache, the ownership is not moved.
3. When the block is replaced from the owner's cache, it should be written back to the shared memory if it is the dirty state. After the replacement, the shared memory is the owner of the block.
4. When the processor which the cache is attached write to a block, the cache become the owner of the block.

The state transition of the block is shown in Figure 2. Unlike the original Keio protocol, the update protocol is also supported. The attribute which specifies a protocol is attached to the page table. Thus, either of the invalidate or update protocol can be selected for each page. The ownership which was not supported in the original protocol is introduced. Unlike Berkeley protocol[3] which also provides the ownership, the ownership mechanism in Modified-Keio protocol is used to minimize the write backs to the shared memory. Like the original Keio protocol, transactions for message transfer without using

the shared memory are also supported (It is not be shown in the diagram to avoid the confusion).

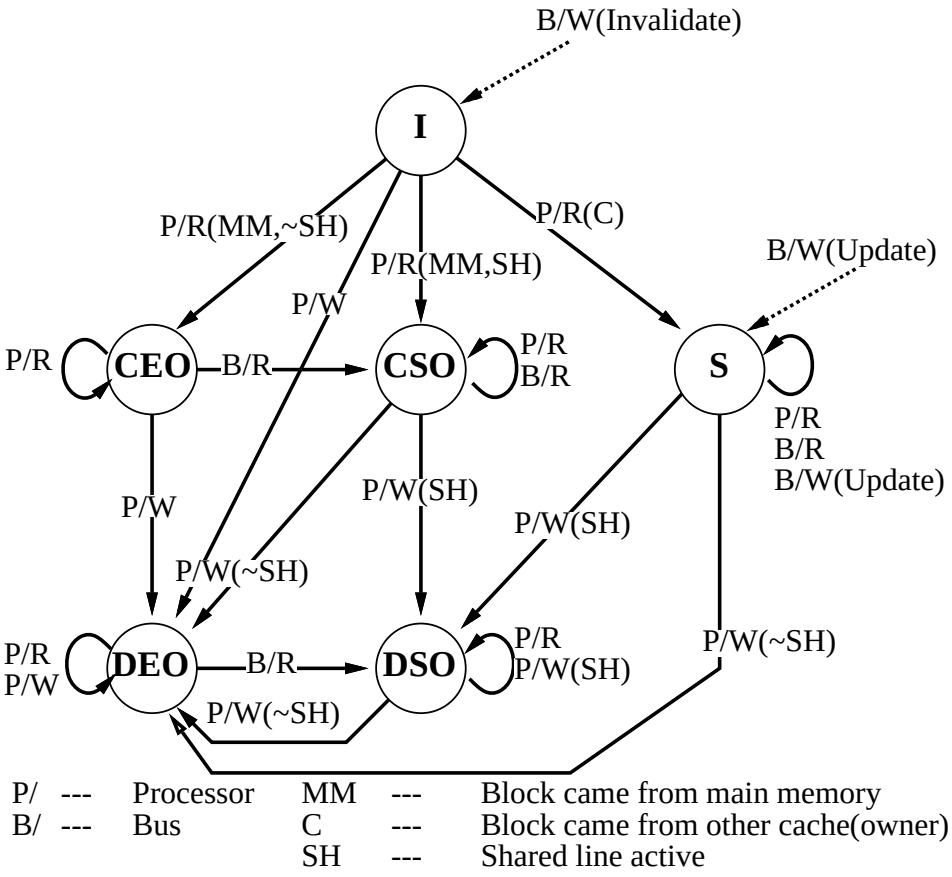

Figure 2: State transition diagram of Modified-Keio protocol

3 Combining the synchronization mechanism

In most bus connected multiprocessors, the synchronization is performed using shared variables on the shared memory. Atomic synchronization operations such as Test&Set and Fetch&Φ[5] are supported usually in the shared memory controller. However, the synchronization on the shared memory increases the communication between the shared memory and the cache, and thus, may degrade the performance in the WSI implementation.

Therefore, we propose a synchronization mechanism combined with Modified-Keio protocol. In this method, a variable for synchronization (here, variable X) is allocated to each page whose attribute is the update protocol. Only one synchronization variable is allowed to be allocated on each cache line. Two additional tags: write interrupt flag and zero interrupt flag are associated to each cache line for inter-processor interrupt mechanism.

Here, we prepare the Fetch&Dec(X) operation, which is a synchronization operation belongs to the class of Fetch&Φ. Figure 3 shows the operation of the Fetch&Dec(X).

Each cache controller equips a hardware adder for decrement operation, and supports this operation in the distributed manner. If X is not zero, the value of X is fetched and **indivisibly** decremented. The cache controller locks the bus first, then, it returns X to the processor, and at the same time, X-1 is sent to the bus through the decrementing hardware.

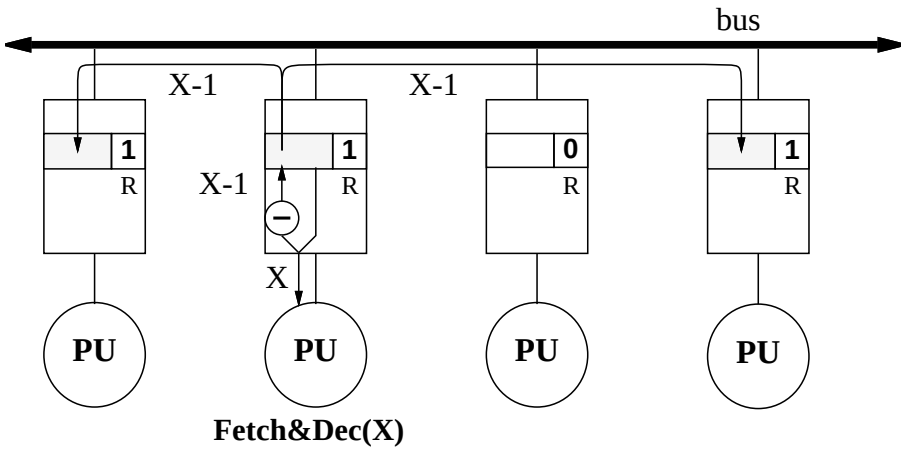

Figure 3: Fetch&Dec(X) operation

Unlike usual Fetch&Dec operation, if X has been zero, it never be decremented no longer, and just the value zero is obtained by this operation. In this case, the bus is not used. Therefore, if the processor is in the busy-waiting state for X, the bus is not influenced. This mechanism is easily combined to the cache controller with only addition of decrementing hardware and a bus wire which indicates the synchronization operation. It never use the shared memory unless the cache line used for the shared variable is written back.

This mechanism is also used for sending interrupt requests between processors. When a processor executes the write operation into variable X, the interrupt request is sent to all processors which holds the cache line and associated write interrupt flag is set. The zero interrupt flag is used for interrupt request caused by the Fetch&Dec(X) operation. If X becomes zero as the result of the Fetch&Dec(X) operation, interrupt signals are sent to processors whose associated zero interrupt flag has been set. Many useful synchronization operations, including the barrier synchronization, message multicast and counting semaphore can be easily implemented with this mechanism. When this inter-processor interrupt mechanism is utilized, if the cache line for a synchronization variable is written back, the interrupt flags must be saved. Therefore, such write back operation causes an interrupt to processor which the cache is attached.

4 Evaluation

To evaluate the efficiency of Modified-Keio protocol, we used the multiprocessor simulator MILL (Multiprocessor Instruction Level simuLator)[6]. MILL is implemented on a Sun workstation using the light-weight processes package. Every instruction of a

processor in the target multiprocessor is directly interpreted, and other components of the multiprocessor such as caches, shared memory and bus are also fully simulated by software. The current target multiprocessor of MILL is a bus connected multiprocessor ATTEMPT-0[4]. In this machine, each processor with off-chip snoop cache is connected to a shared memory through the bus.

Two application programs which are executed on the target multiprocessor are used for evaluation.

- **MP3D** is one of SPLASH benchmark suite[7]. It solves a problem in rarefied fluid flow simulation. Monte Carlo method is used. It simulates the hypersonic flow of idealized diatomic molecules in a rectangular tunnel with openings at each end and reflecting walls on the remaining sides. The object we used is a single flat sheet placed at an angle to the free stream of the molecules (`test.geom`, accompanied with the program). The simulation was executed 100 time-steps with 400 molecules.

- **Logique**[8] is a parallel logic circuit simulator based on a query algorithm in Chandy-Misra's discrete event simulation algorithms. In Logique, the query is implemented as an access to the shared memory for the efficiency. The state of each element in the circuit, and message queues are placed on the shared memory. And the schedule list for the simulation is placed on the local memory. The target circuit we used for the simulation is 4bit ALU.

For comparison, following 5 protocols are used.

- Modified-Keio : write-invalidate for synchronization variables, write-update for other shared data.

- Illinois : write-invalidate

- Berkeley : write-invalidate

- Dragon[3] : write-update

- Illinois – Dragon : Dragon for synchronization variables, Illinois for other shared data.

cache size	32Kbytes
line size	16bytes
set associativity	4way

Table 1: Simulation parameters

Logique uses the special synchronization mechanism of ATTEMPT-0 which is also supported by MILL. So, Modified-Keio uses only the invalidate protocol, and Illinois-Dragon combination is not used for evaluations. Table 1 shows the simulation parameters. Since it is difficult to implement large cache memories in a multiprocessor chip, the size of the cache memory is selected to be smaller than current multiprocessors.

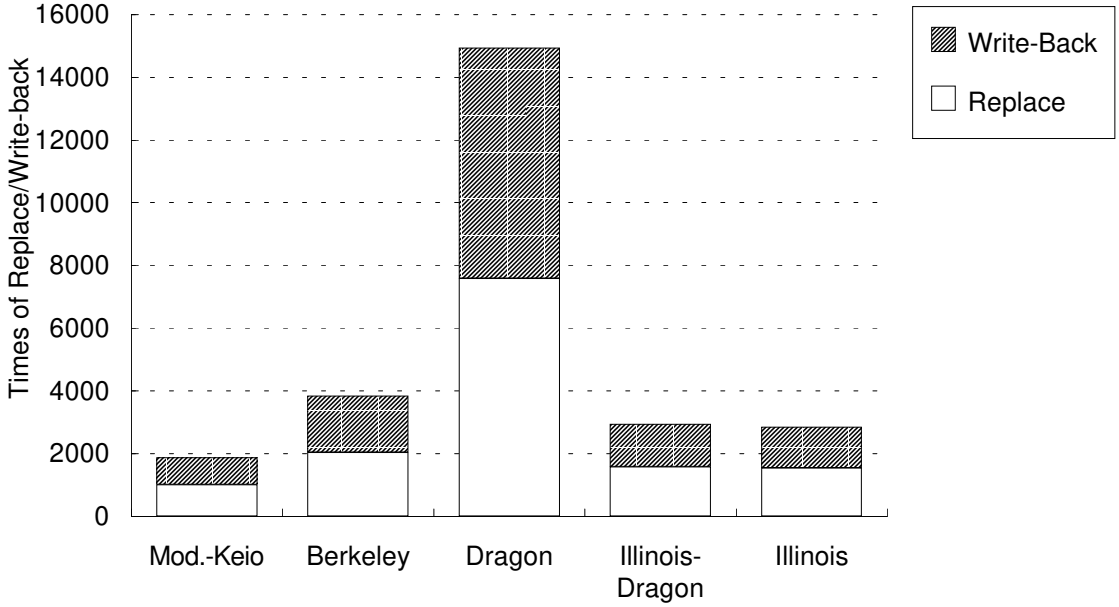

Figure 4: Number of replaces, write backs (6PU, MP3D)

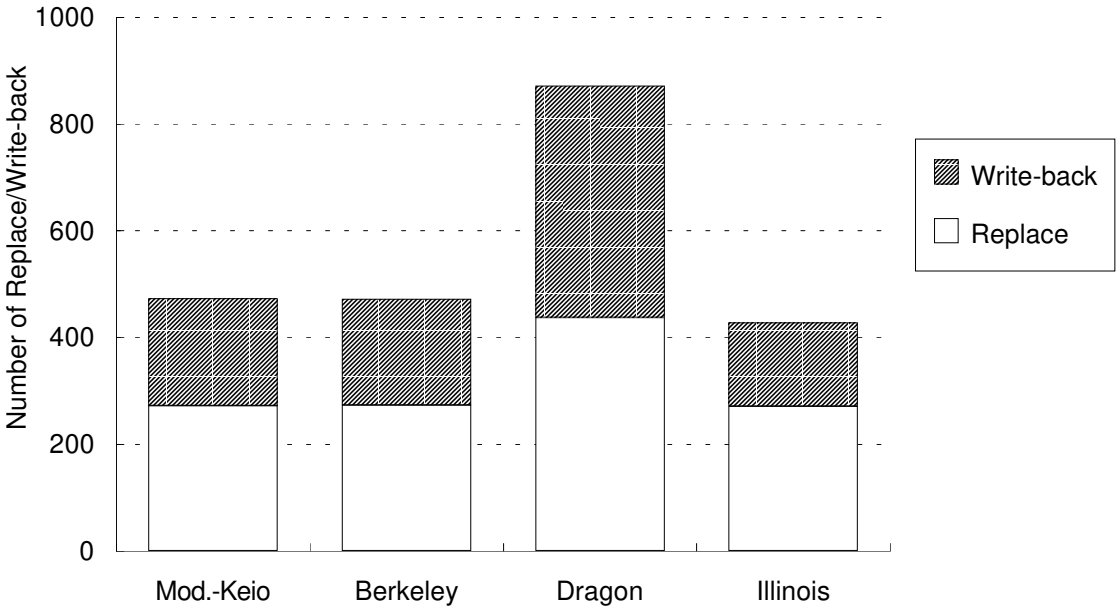

Figure 5: Number of replaces, write backs (6PU, Logique)

Figure 4,5 shows the number of replaces and write backs to the shared memory. Since the capacity of the cache memory is small, many replaces were observed. Especially for the Dragon, the number of replaces is large because the lines with dirty-shared state remain in the cache memory.

In MP3D, Modified-Keio shows the smallest number of replaces and write backs. While Illinois shows the smallest number in Logique, it does not include the write backs which are required for the transfer of a dirty line. If it is included, Modified-Keio shows the smallest number of replaces and write backs.

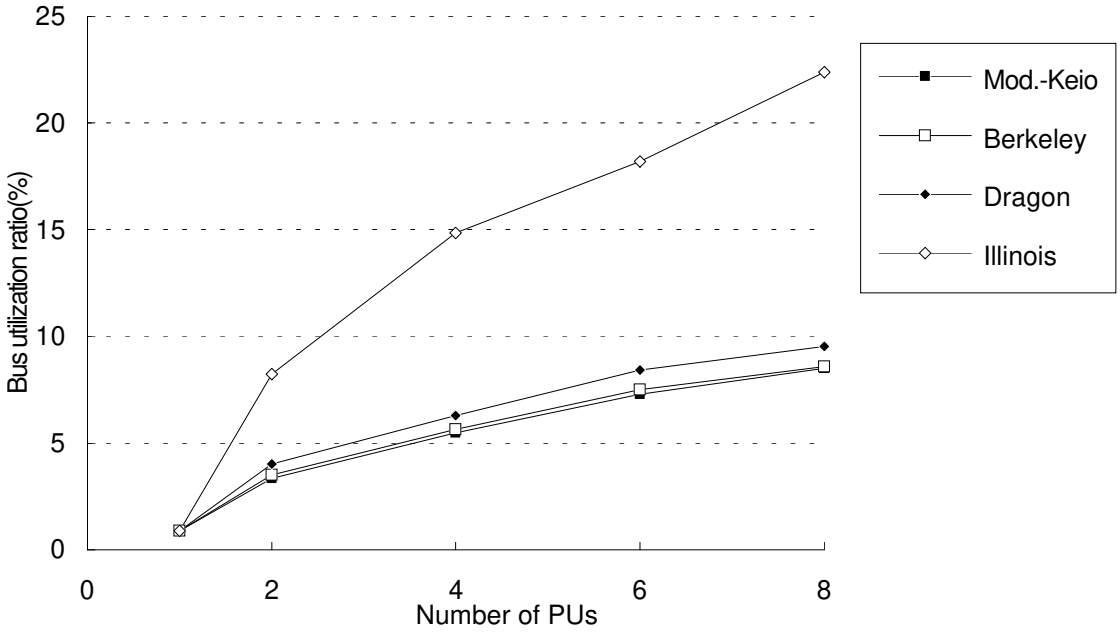

Figure 6: Bus utilization ratio (Logique)

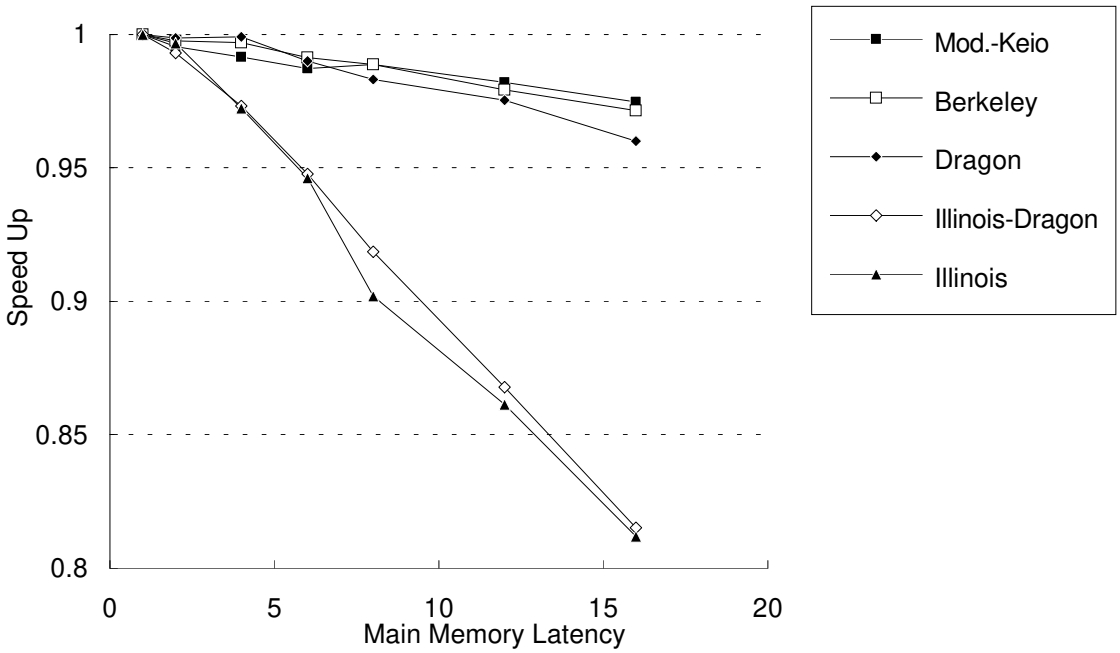

Figure 7: Shared memory latency v.s. speedup (4PU, MP3D)

Figure 6 shows the bus utilization ratio of Logique when the number of the processors is increased. Modified-Keio shows the lowest bus utilization ratio. Since Illinois needs a write back of the line to the shared memory when a dirty line is transferred between caches, its bus utilization ratio is large. Figure 7 shows the performance degradation of MP3D with 4 processors when the shared memory latency is increased. Illinois and Illinois-Dragon show large performance degradations. In the other 3 protocols, the performance degradation is small. Modified-Keio achieves the best performance.

5 Related works

In the current state of the technology, most researches focus on the WSI or on the multiprocessor chip implementation itself [9][10]. There are few researches on the cache architecture dedicated for the WSI or on-chip multiprocessor implementation. Mori assumes a high performance shared bus connected multiprocessor on the WSI implementation[11]. In his research, the fault tolerant mechanism is mainly considered, and no cache is attached.

In this paper, the snoop cache mechanism for the WSI or on-chip multiprocessor implementation is discussed. However, there are other two candidates:

- Shared cache
  In the recent multiprocessors, shared cache is rarely used because of the cache contention problem. However, if dual/triple/quadruple port memory is easily used on the WSI or on-chip environment, the shared cache approach is feasible. Unlike the snoop cache, there is not duplicated data on the shared cache. Therefore, when the size of the total cache memory is limited, this approach is advantageous.

- Crossbar or multiple buses
  In the WSI or on-chip environment, high bandwidth crossbar or multiple shared buses can be used because there is no pin-limitation problem for connecting processors. However, if the crossbar or multiple bus is used for the connection with the off-chip shared memory, the pin limitation problem of the chip and the delay for the off-chip communication will degrade the performance. In this case, shared buffer, shared cache or other technique is required.

In the current of the research, the above two approaches are also attractive. The comparison with the snoop cache approach is our future work.

6 Conclusion

In this paper, we propose a snoop cache protocol for the WSI implementation which minimizes the accesses to the shared memory. In Modified-Keio protocol, both write-invalidate and write-update type protocols can be used according to the nature of the shared data. It also supports the simple synchronization mechanism with Fetch&Dec operation and inter-processor interrupt.

Detailed simulation with practical parallel applications shows the efficiency of our protocol. Now we are constructing a prototype machine using R3000 processors to evaluate the actual performance of the protocol.

References

[1] J.R.Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic," Proc. of 10th Int'l Symp. on Computer Architecture, pp. 124-131, Jun. 1983.

[2] T. Matsumoto, "Fine-Grain Support Mechanisms," IPSJ SIG Reports (in Japanese), Vol.89 No.60, ARC-77-12, pp.91-98, Jul. 1989.

[3] J.Archibald, J.-L.Baer, "Cache-Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model," ACM Trans. on Computer Systems, Vol.4, No.4, pp. 273-298, Nov. 1986.

[4] H.Amano, T.Terasawa and T.Kudoh, "Cache with synchronization mechanism," Proc. of 11th IFIP World Computer Congress, pp.1001-1006, Aug. 1989.

[5] A.Gottlieb, R.Grishman, C.P.Kruskal, K.P.McAuliffe, L.Rudolph, M.Snir, "The NYU Ultracomputer – Designing an MIMD Shared Memory Parallel Computer," IEEE Trans. on Computers, Vol. C-32, No. 2, pp. 175-189, Feb. 1983.

[6] T.Terasawa, H.Amano, "Performance Evaluation of the Mixed-protocol Caches with Instruction Level Multiprocessor Simulator," Proc. of IASTED Int'l Conf. on Modeling and Simulation, May, 1994.

[7] J.P. Singh, W. Weber and A. Gupta, "SPLASH: Stanford Parallel Applications for Shared-Memory," Tech. Report, Computer System Laboratory Stanford University, 1992.

[8] T.Kudoh, T.Kimura, H.Amano, T.Terasawa, "A parallel logic simulation algorithm based on query," Proc. of International Conference on Parallel Processing, Vol.III, pp.262-266, Aug. 1992.

[9] P.P.Gelsinger, et al., "Microprocessors circa 2000", IEEE Spectrum, pp.43-47, Oct., 1989.

[10] M. Hanawa, et al., "On-Chip Multiple Superscalar Processors with Secondary Cache Memories", Proc. of ICCD, pp.128-131, 1991.

[11] H.Mori, "Fault Tolerant Architecture for Multi Processors on WSI", IEICE SIG Reports (In Japanese), WSI92-21, pp.250-256, 1992.