Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Evaluations on Memory Buffers for Shared Bus Multiprocessors

Jeng-Ping Lin, Shang-Ching Sue, Shih-Chang Wang, Sy-Yen Kuo, Chin-Sung Chen*, and Hong-Chich Chou*
Department of Electrical Engineering
National Taiwan University
*Computer & Communication Laboratories
Industrial Technology Research Institute

Abstract

This paper evaluates the optimal number of memory buffers we should include in the memory controller to improve the system performance. We focus on shared-bus multiprocessor (MP) systems adopting DRAM (Dynamic Random Access Memory) as the shared memory. In order to evaluate the design tradeoff in various conditions, extensive simulation was conducted by employing a commercial simulation tool. Using the MP system model constructed in the simulation tool, we could evaluate the optimal number of read or write buffers in the memory controllers under different configurations. In addition, it could even combine the model with trace-file under a slight modification. By adding optimal number of read and write buffers, the system performance is shown to increase efficiently and significantly.

Keywords: Multiprocessor, memory buffer, simulation, superscalar, address pipeline.

1.Introduction

Although the computational speeds of conventional uniprocessors have increased dramatically since the first vacuum tube computers, there is still a demand for even faster computing power which is far in excess of what can currently be supplied. Many of the proposed solutions involve the construction of *MP*(multiprocessor) systems which provide a shared-address space, allowing individual memory accesses to be used for communication and synchronization. Many research projects on such *MP* systems have been presented in the literature. *MP* systems are used today to provide better performance with low-cost and high-performance microprocessors.

Currently, most commercially available *MP* systems are based on the shared-memory shared-bus architecture[1-2] (see Fig. 1) Such *MP* systems are popular for two reasons:

1) the shared-bus interconnect is easy to implement[3]
2) the shared-bus interconnect allows an easy solution for cache coherence problems[4-5]

We are currently involved in a project to build a high performance *MP* workstation. The proposed *MP* system adopts the architecture in Fig. 1 and using PowerPC 620 as the processors.

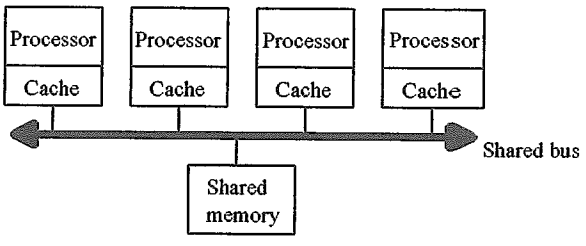

Fig. 1: Shared-memory *MP* architecture.

We evaluate the optimal number of memory buffers, including *read* buffers and *write* buffer in the memory controller under different configurations. An *effective*

read or write buffer is defined as a buffer which will affect the system performance if added to the memory controller. By using the *SES/workbench* simulation tool [6], we show that the performance of the MP system is improved significantly by the utilization of memory buffers and get the optimal number of buffers in the target system.

The remainder of this paper is organized as follows. The motivation and background are given in Section 2. In Section 3, we discuss the approach for the improvement of the system performance by adding buffers in the proposed *MP* system. In Section 4, the processor execution model and the system model are addressed. In Section 5, we evaluate the maximum number of effective memory buffers. In Section 6, we use *SES/workbench* to construct the simulation environment for the proposed *MP* system and summarize the simulation results. Finally, we have the concluding remarks and future works.

2.Motivation and background

MP systems have emerged as a key enabling technology in modern computers, driven by the ever-increasing demand for higher performance, lower cost, and sustained productivity in real-life applications. From 1970 to 1993, the progress in design technology of semiconductor chips has made the system performance improved by 30% each year. With the inclusion of parallel processing technology, the system performance has increased by at least 70% each year now.

The processors used in the proposed *MP* system are PowerPC 620's. PowerPC 620 is a highly integrated single chip microprocessor. The major differences between conventional RISC processors and PowerPC 620 processors are the superscalar architecture and the high performance bus interface which supports the MP system[7-8].

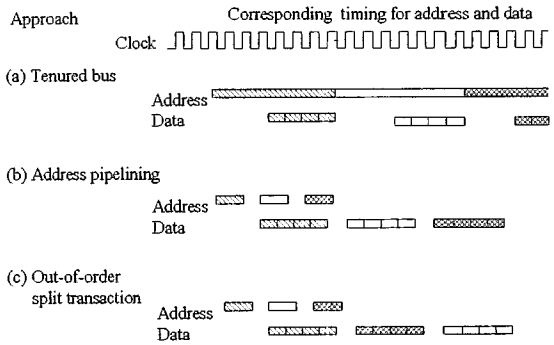

Fig. 2: Bus utilization for three well-known shared-bus implementations.

The PowerPC 620 splits its bus into the address bus and the data bus. In Fig. 2, we show three well-known shared-bus implementations and the corresponding timing charts for the address and data.

The conventional approach (see Fig. 2(a)) employs a tenured bus in which the memory latency governs the use of the address because the transfer of address and data are multiplexed on a common bus. To improve the bus performance, we could split the common bus into separate buses so that *address-only* operations can be efficiently overlapped with data-transfer operations. This is known as address pipelining (see Fig. 2(b)). Another further enhancement of the bus utilization is to support out-of-order transactions (see Fig. 2(c)), that is, the data responses do not have to be returned to the requesting processor in the order that they came out. The bus utilization for the proposed PowerPC 620 *MP* system is based on the split transaction method in Fig. 2(c).

The more detailed view of the proposed *MP* system architecture based on PowerPC 620 is shown in Fig. 3. The overall performance is heavily influenced by the design of the local cache of each processor, the shared-bus interconnect and protocol, and the shared memory. The large latency of memory accesses is one of the major impediments to achieve high performance in shared-memory MP systems. Adding additional buffers

is an attractive technique for hiding this latency by allowing the overlapping of memory accesses with other computations and memory accesses. In the next section, we will present the proposed approach for adding additional buffers in the MP systems.

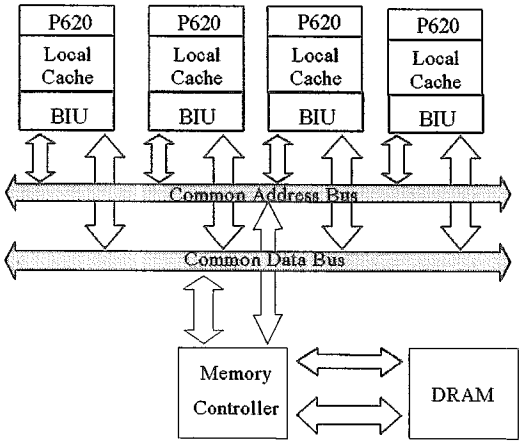

Fig. 3: *MP* system overview.

3.Adding buffers in the MP systems

There are two places for adding memory buffers in Fig. 3. One is between the processor and the bus, i.e., in the *BIU (see* Fig. 4), and the other is between the bus and the shared memory, i.e., in the memory controller (see Fig. 5). In this paper, buffers in the *BIU* are in the processor and thus are fixed so that we would not evaluate on them. Instead, we evaluate the effect of memory buffers in the memory controller.

PowerPC 620 has implemented several memory buffers in the *BIU*. There exist two read buffers, three write buffers, and one *intervention* buffer which has the highest priority. Memory buffers in the *BIU* are similar to memory buffers in the memory controller.

The *intervention* operation on bus has become a popular technique to reduce the read latency. The definition of *intervention* is that when a read miss or write miss bus operation occurs and some processor holds the modified data, the modified data will be supplied by the holding processor. A processor will issue bus read requests when cache misses occur. The required data can be sourced from the memory and this is called "***memory reply***". It can also be sourced from another processor and this cache to cache transfer is called "***intervention***". Whenever a read request is acknowledged by reading data from a processor's cache, the memory also accepts the cache read response as a sharing writeback. Therefore, we also include the intervention in our model to make it more realistic.

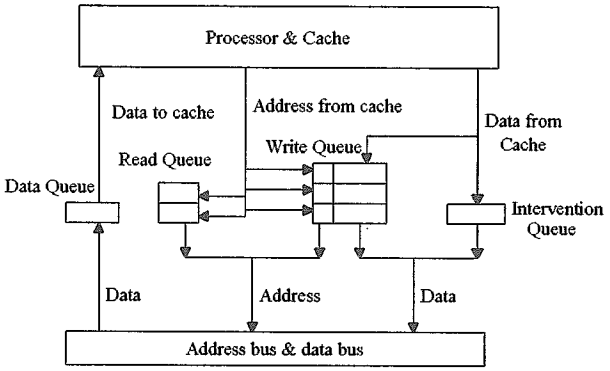

Fig. 4: Memory buffers in the *BIU*.

In Fig. 5, *RAB*, *WAB*, and *WDB* are added to buffer the data from the processors to the shared memory. On the other hand, *RDB* are added to buffer the data from the shared memory to the processor. Further, we also use *IDB* and *IAB* to buffer the *intervention* data which is used to update the shared memory mentioned earlier.

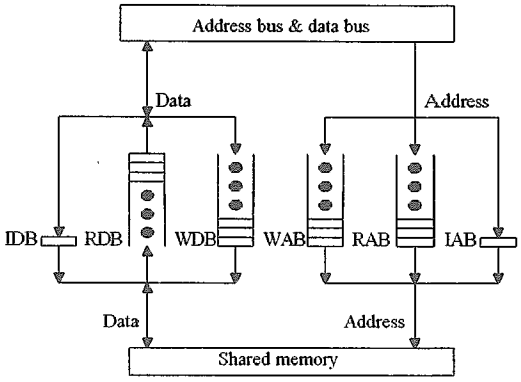

Fig. 5: Adding memory buffers in the memory controller.

RAB	*read* address buffer
RDB	*read* data buffer
WAB	*write* address buffer
WDB	*write* data buffer
IAB	*intervention* address buffer
IDB	*intervention* data buffer

4.Processor execution model and system model

We use a simplified processor model and assign various probabilities to the instructions. If an instruction is blocked and can not be issued from the instruction dispatch unit, the processor is idle and the idle period is called "processor waste time" Only cache-miss bus requests or bus control requests to the *BIU* can appear in the shared bus.

Merely from cache miss and invalidation operations, we can not tell a blocking request from a non-blocking request for superscalar processors. The potential blocking requests are called ***B-requests***. Whether the processor can issue another *B-request* if there is already an outstanding *B*-request issued by the same processor depends on if the processor can distinguish different types of read requests, for example, the *data load* request and the *instruction fetch* request. These two situations are also considered in the proposed *MP* system. We construct the complete system model according to the bus behavior of PowerPC 620 (see Fig. 6).

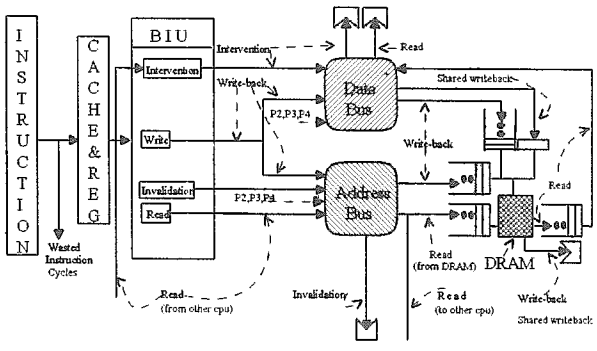

Fig. 6: System model.

We assume that the address and data bus arbitration mechanisms are fair, e.g., the bus arbitration adopts a *FCFS* strategy. If there are several bus requests which are already received but not granted by the address bus arbiter, a newly arriving request after them will wait for their services by the address bus. After the request is serviced by the address bus, the request enters the memory buffers or wakes up another processor to intervene the data. If the bus is busy servicing a request while the arbitration for mastership for the next request is taking place, the arbitration time is overlapped completely with the service time and does not contribute to the time spent by a request on the bus. After the detailed description of the proposed processor execution and the system models, the evaluation is described in the next section.

5.Evaluation of the number of buffers

For the proposed design of memory buffers, the system performance can be affected by the memory buffers when some specific bus request patterns occur We observe the characteristics of bus requests and evaluate the number of memory buffers. The simulation part is presented in Section 6, and it can be used to justify the evaluation here.

There is a model called *CMVA (Customized Mean Value Analysis)* proposed in the literature[9-10]. It assumes that the average task characteristics on each processor are the same and the relationships among the mean values of the derived times are stable and consistent. It is clear that the uniform distribution assumption is not reasonable, because the applications usually have non-uniform bus request traces. Moreover, it is difficult to obtain accurate analytical equations. Consequently, our main goal is on evaluating the maximal number of effective memory buffers by a simplified analytical analysis and discussing how the read and writeback requests affect the system performance.

The following parameters are needed for later discussions:

TA	the address bus access time of a bus request.
TD	the data bus access time of a block transfer either for a cache write-back or a main memory reply.(only burst mode transfer considered)
WA	the waiting time of the common address bus.
WD	the waiting time of the common data bus

Firstly, we consider the worst case that all *N* processors issue a read request respectively and the corresponding data are not intervened by another processor and the time (*WD+TD*) is large enough for *N* read requests to enter *RAB* in the memory controller. Then the maximal number of effective *RAB* in the memory controller is *N* because a processor cannot issue another read request if the previous read request is still outstanding. The above case is true if the processor can not distinguish two outstanding read requests. If the processor can distinguish two outstanding read requests, the maximal number of *RAB* in the memory controller becomes *2N*. Despite the invalidation may use *RAB*, the service time is so short according to the bus behavior in the proposed system that the maximal number of effective *RAB* is less than 2. *WAB/WDB* is dependent on the write-back requests. When a processor issues write-back requests to its *BIU*, ideally these requests will not block the processor's computation. But if the time (*WA+TA*) is too large and there are not enough WAB/WDB in the *BIU*, the next write-back requests can see the previous write back request and the processor will be blocked due to the fact that the WAB/WDB in the *BIU* is full. However, the *WAB/WDB* in the memory controller can be used to eliminate this possibility of blocking the processor. The maximum number of effective *WAB/WDB* in *BIU* depends on the maximum number of write back requests which the processor can distinguish among them. If there are m *WAB/WDB* buffers in the *BIU*, the maximal number of effective *WAB/WDB* in the memory controller is *mN* under the assumption that the processor has instructions like *Flush* and can only distinguish their own *m* write back requests. Notice that the probability of replacement or *flush* is also a factor of *WAB/WDB*. Therefore, the maximum number of *WAB/WDB* in the memory controller depends on the maximum number of *WAB/WDB* in *BIU*.

As to the maximal number of effective *RDB*, we know that it could buffer the data from the memory to the processor. As mentioned earlier, the maximum number of effective *RAB* in the memory is *N* or *2N*, depending on whether the processor can distinguish two different read requests. The maximal number of effective *RDB* is equal to the maximal number of *RAB*, since the data accessed by the addresses in *RAB* enters into *RDB*.

Although the maximal number of effective *RAB* in the memory controller is only a function of *N*, the optimal number of *RAB* in the memory controller is also affected by the *WAB/WDB* in the memory controller. Since for the memory controller, there is no preference given to the requests in *RAB* or the requests in *WAB/WDB*, the read requests and the write-back requests enter the memory one by one in turn. Thus, the requests in *WAB/WDB* will delay the requests in *RAB*. There are many feedback interactions among these parameters. Although it is hard to get a function that explicitly represents their interactions, we know there are two main factors that affect the system performance. With increasing number of *RAB*, the possibility that read requests would stay in the *BIU* and thus block the processor will be greatly reduced. The result is an improved system performance. On the other hand, with increasing number of *WAB/WDB*, the first one is the same as the case of *RAB* and has a positive impact. Secondly, the larger queue of *WAB/WDB* means that more write requests could stay in these buffers. This means that a read request must now wait for more requests to be finished. Consequently, the response time

of a read request becomes longer and greatly increase the blocking probability. The net effect is underestimated. By observing the simulation result, we can obtain the result of their dependency.

6. Simulation by SES/*workbench*

This paper builds a MP system model using *SES/workbench*[6] which could develope of structured models using multiple layers and hierarchy. In *SES/workbench*, it sonsists of transactions and various kinds of resource nodes. The transactions would flow between nodes according to the specified course and could be used to represent the instructions. The resource nodes could represent the system resources such as the address bus, the data bus, read buffers, write buffers, and so on. The advantages are that you could specify the services strategies (such as FCFS) for the resource nodes to service the requests. By using *SES/workbench*, we could simulate the behavior of MP systems more realistically and easily. Moreover, due to the modularity and capability of SES/workbench, we could combine our model with the trace-driven method under a little modification.

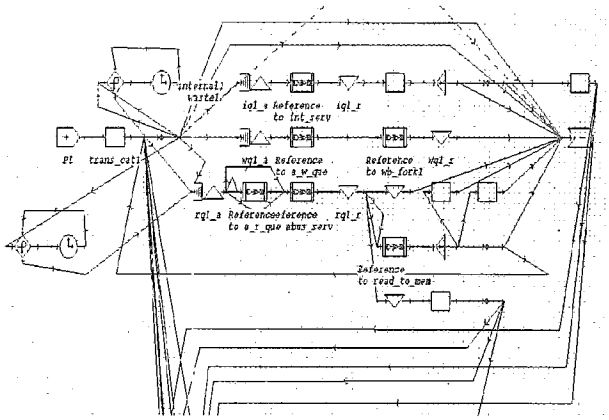

Fig. 7: The transaction path for one processor.

From Fig.7, the complete simulation model is built for the proposed *MP* system using split buses based on PowerPC 620 and memory buffers. Under the various probability distribution, the averaged wasted time of a processor in cycles will be taken as the performance measure. We take a system with 4P (four processors) as the simulation base because it is typical in commercial markets for shared bus *MP* systems. However, the same methodology can also be applied to systems with more than four processors connected to the shared bus.

We observe the processor-waste-time in cycles, the number of *RAB* and the number of *WAB/WDB* (see Fig. 15-20) under different memory latencies assuming that the processor cannot distinguish two outstanding read requests. In the simulation, bus clock cycles are 2 *PClks (Processor* Clock cycles) and hit rate is 98%. Each processor has 200,000 cycles to issue instructions. The processor-waste-time is summed over four processors. Due to this high hit rate, the number of bus requests will be small. The difference among different numbers of memory buffers in the simulation is not obvious if we look at the percentage of the processor-waste-time over the total simulation cycles. Thus, we observe the real value of processor-waste-time rather than its percentage.

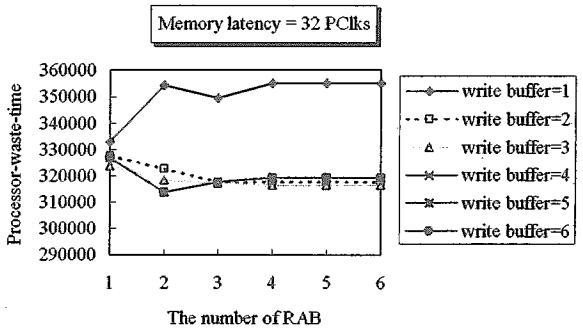

Fig. 8: The effect of read buffers.

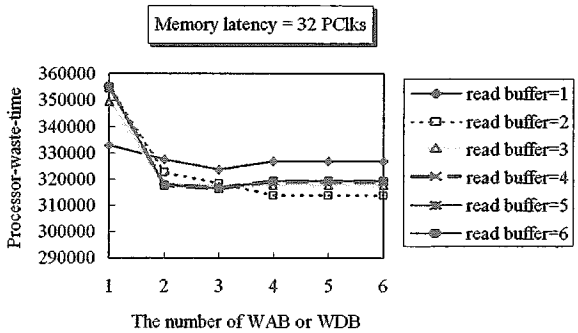

Fig. 9: The effect of write buffers.

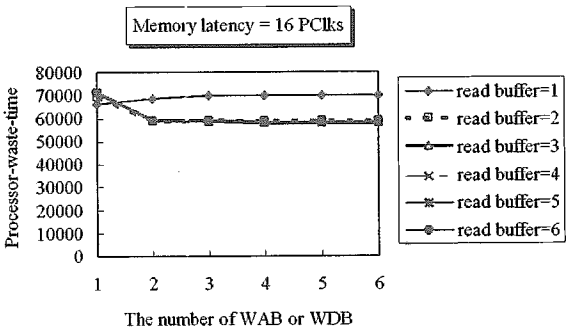

Fig. 10: The effect of read buffers.

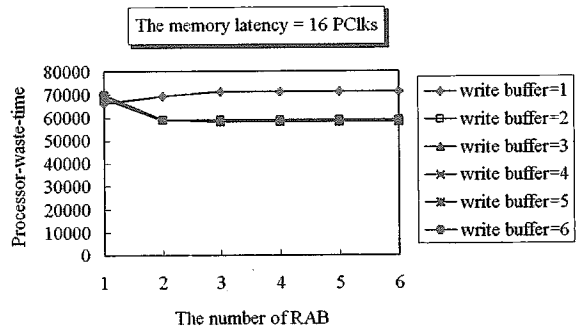

Fig. 11: The effect of write buffers.

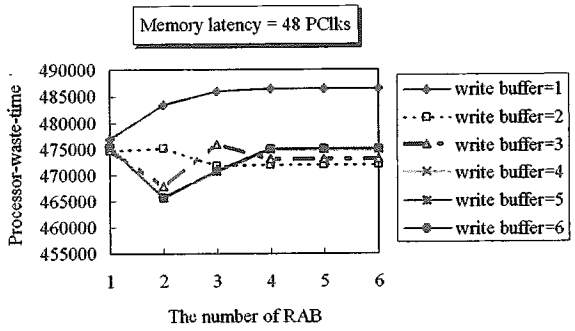

Fig. 12: The effect of read buffers.

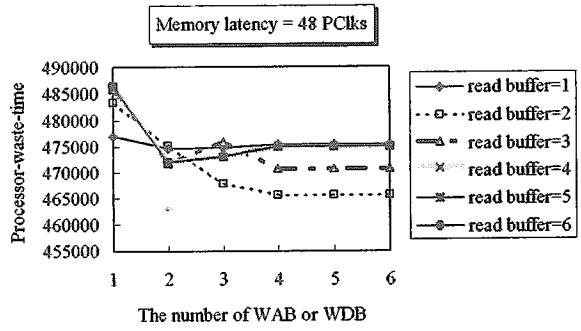

Fig. 13: The effect of write buffers.

From the above figures, it is clear that the processor-waste-time becomes saturated after adding a few buffers. With longer memory latency(Memory latency = 32 or 48 *PClks*), it saturates at four. On the other hand, with shorter latency (Memory latency = 16 *PClks*), it saturates at two. There exists an optimal combination of read and write buffers which results in the lowest processor-waste-time for different memory latencies. When the memory latency is 32 *PClks*, the optimal combination is 2 read buffers and 4 write buffers. When the memory latency is 16 *PClks*, the optimal combination is 2 read buffers and 2 write buffers. When the memory latency is equal to 48 *PClks*, the optimal combination is the same as the case when the memory latency is 32 *PClks*.

As the memory latency is decreasing, the optimal number of buffers is decreasing. This is because the purpose of the buffers is to overlap the bus arbitration time with the longer memory latency . Although we know that the more the read buffers, the less the processor-waste-time, the requests in write buffers will reduce this effect because the service time of read requests is delayed. Therefore, the faster the memory, the less the counterpart effect. Such a dynamic phenomenon is hard to discuss, but we could see the net effect by simulation. For the case that the processor can distinguish two outstanding requests and *flush* operations are considered(the figures are not shown), we find that the number of read buffers is saturated at 5, not 8(i.e., $2N$), and the number of write buffers is saturated at 7, not 12(i.e., mN). This is because the alternative pattern for two different types of outstanding read requests does not occur frequently and the probability of *flush* operations is not high. With the same methodology, we can analyze the case for different *BIU* of the processor and different numbers of processors connected to the shared bus. Even various bus behaviors can be easily constructed by the *SES/workbench* to get a more realistic result.

From the above observation, we conclude that the optimal number of *read* or *write* buffers in the memory

controller is not necessarily the maximal number of effective read or write buffers in the memory controller. And the maximal number of effective read buffers in the memory controller is at most *N* or 2N (N is the number of processors connected to the shared bus), depending on whether the processor can distinguish the different types of read requests or not, such as *data load* and *instruction fetch*. The maximal number of effective write buffers in the memory controller is at most *mN*, where *m* is the number of write buffers in the *BIU* of the processor. By adding the optimal number of read and write buffers, the system performance is shown to increase significantly.

7.Conclusions and future works

Adding memory buffers is a good idea to buffer the concurrent bus requests to the memory and to buffer the data to the requested processor. We determine the optimal combination of read and write buffers to achieve higher system performance efficiently by a simulation tool, the *SES/workbench*. The intervention idea is also introduced in the proposed *MP* system to increase the performance of the system and the corresponding buffer problem is solved by introducing an intervention buffer.

We take a simple model tool *SES/workbench* instead of the complicated trace driven method. In addition, we concentrate on the effect of memory buffers which are affected by the bus requests issued by the processor. The bus protocol and cache states are ignored. From the simulation result, it justifies the evaluation by the proposed processor executing timing model. With the flexible and maintainable models provided by *SES/workbench*, the construction time of the proposed *MP* model is reduced and we could get more realistic simulation results.

As to future work, if the synchronous *DRAM* with the pipelining capability is adopted, it could be modeled as follows. If the *SDRAM* has three stages of pipelines, the processing time of the read buffer and write buffer would be equal to that of the first stage of *SDRAM*. Further, we could combine *RAB* with *WAB* into a single address buffer in the memory controller, an additional hardware such as the reordering mechanism is needed. We must keep two pointers to distinguish between the read requests and the write-back requests. Although this complicates the hardware design, it reduces the waste of the address buffers. Based on the same evaluation method, we will evaluate these effects in the future.

References

[1] C. D. Rose, "Encore eyes multiprocessor market", *Electronics*, July 8, 1985.

[2] Sequent Computer Systems, Inc., "Balance 8000", *Technical summary*, Nov. 1984.

[3] A. Hopper, A. Jones, D. Lioupis, "Multiple vs wide shared bus multiprocessors", in *Proc. 16th Annu. Symp. Comput. Architecture*, pp. 300-306, June 1989.

[4] J. Archibald, J. Baer, "An evaluation of cache coherence solutions in shared-bus multiprocessors", *ACM Trans. Computer Systems*, pp. 273-298, Nov. 1986.

[5] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," in *Proc. 10th Annu. Symp. Comput. Architecture*, pp. 124-131, June 1983.

[6] SES/workbench User's manual and Reference's manual Release 2.1. Feb. 1992.

[7] E. Silha and G. Paap, "PowerPC: A performance architecture", *Proceedings of COMPCON*, pp. 104-108, Feb. 1993.

[8] IBM Confidential, "PowerPC Implementation Features", *Book IV*, 1994.

[9] S. Lentenegger and M. K. Vernon, "A mean value performance analysis of a new multiprocessor architecture", in *Proc. ACM SIGMETRICS Conf. Measurement and Modeling of Comput. Syst.*, May 1988.

[10] M. K. Vernon, R. Jog, and G. S. Sohi, "Performance analysis of hierarchical cache-consistent multiprocessors," *Perform. Eval* vol. 9, pp. 287-302, 1989.