

# Request Reordering to Enhance the Performance of Strict Consistency Models

YoungChul Sohn, NaiHoon Jung, SeungRyoul Maeng  
Dept. of EECS, Korea Advanced Institute of Science and Technology, Korea  
{ycsohn, nhjung, maeng}@calab.kaist.ac.kr

**Abstract**—Advances in ILP techniques enable strict consistency models to relax memory order through speculative execution of memory operations. However, ordering constraints still hinder the performance because speculatively executed operations cannot be committed out of program order for the possibility of mis-speculation. In this paper, we propose a new technique which allows memory operations to be *non-speculatively* committed out of order without violating consistency constraints.

**Keywords**—multiprocessor, memory consistency model, ILP

## I. INTRODUCTION

THE memory consistency model is a crucial factor for the performance of shared memory multiprocessor systems. Strict consistency models (such as sequential consistency (SC) or total store ordering (TSO)) offer intuitive programming interface, but the inability to perform memory operations out of program order limits the performance.

Modern microprocessors incorporate techniques to exploit instruction-level parallelism (ILP). ILP techniques enable aggressive optimizations for strict consistency models, which relax memory order speculatively. Gharachorloo et al. [2] proposed hardware prefetch and speculative load execution. These two optimizations significantly improved the performance of strict consistency models through issuing memory operations out of program order. However, ordering constraints in strict consistency models still hinder the performance [6]. First, the store-to-load ordering (in SC) prohibits a load from bypassing prior stores and retiring from the reorder buffer. It may cause a high latency store to block the instruction flow through the reorder buffer. Second, the store-to-store ordering (in SC or TSO) forces stores to be performed one after another. It may cause underutilization of memory units and cache ports.

To alleviate above problems, speculative retirement [6] and SC++ [3] are proposed. Those techniques allow memory operations to be speculatively *committed*<sup>1</sup> out of program order. Thus, memory operations no longer stall processor pipelines waiting for the completion of prior operations. However, out-of-order commitment may incur a consistency violation. To recover from possible consistency violations due to speculative commitment, a processor should store the

architectural state into an additional history buffer and roll back when a mis-speculation is detected.

The effectiveness of the speculative commitment techniques is mainly limited by the size of a history buffer; the number of instructions can be committed speculatively at a time. The size of a history buffer should scale with the performance gap between processor and memory subsystem. To hide the higher memory latency, a processor should provide the larger history buffer [3]. Especially, in case of a speculative commitment of a load, the history buffer should keep rollback information not only for the load itself but also for all of the instructions following the load. Another defect of speculative mechanisms is that previous architectural state should be loaded and updated atomically. This read-modify-write operation may increase contention for the resources. Lastly, frequent rollback due to the early prefetch or false sharing would result in performance degradation [6].

In this paper, we propose a new mechanism, the request reorder buffer (RRB) technique, to alleviate the impact of the store-to-load and store-to-store ordering constraints in strict consistency models. The RRB technique enables *non-speculative* commitment of memory operations (both loads and stores) bypassing prior stores. To guarantee consistency constraints, the RRB technique rearranges the global order of memory operations by delaying a cache coherence request. Because the RRB technique does not require rollback, long memory access latencies can be hidden with small cost of storage, and the negative effects of the speculative techniques can be removed.

## II. THE RRB ARCHITECTURE

This section describes the RRB technique. We assume a cc-NUMA system similar to SGI Origin2000 [4]. Processors incorporate hardware prefetch, speculative load execution, and store buffering. Also, an invalidation-based cache coherence protocol is used. We will describe how the RRB technique works in SC, and it can be directly applied to TSO.

### A. Delaying Coherence Request with the RRB Technique

In the sample program in Fig. 1, suppose that P0 executes the operation *b* before *a* and sets r1 to 0. In this situation, the out-of-order execution of *b* may or may not violate sequential consistency; if the result of *b* is the same as the result produced by an in-order execution of *b*, it does not violate sequential consistency. Otherwise, the out-of-order execution may cause an inconsistency.

Manuscript submitted: 13 Sep. 2002. Manuscript accepted: 22 Oct. 2002. Final manuscript received: 28. Oct. 2002.

<sup>1</sup> We call an operation is *committed* when it updates the processor and memory state; a load is committed when it update destination register and is retired from the reorder buffer, and a store is committed when it updates the cache and is removed from the store buffer (or memory queue).

Initially, A=B=0  
P0: Store A,1 (a)      P1: Store B,1 (c)  
Load r1,B (b)          Load r2,A (d)

Fig. 1. The sample program.

For example, suppose that the operations are executed in the order  $(b, a, c, d)$ . In this case, the in-order execution of  $b$  after  $a$ ,  $(a, b, c, d)$ , would produce the same result,  $r1=0$ , as the out-of-order execution. Thus, the execution  $(b, a, c, d)$  is sequentially consistent. However, the out-of-order execution sometimes leads to an inconsistency. If  $c$  and  $d$  are performed between  $b$  and  $a$ ,  $(b, c, d, a)$ , the result of  $b$ ,  $r1=0$ , is different from that can be obtained by the in-order execution  $(c, d, a, b)$ , which produces  $r1=1$ . In this case, the out-of-order execution violates sequential consistency; the result,  $r1=0$  and  $r2=0$ , can never be obtained in sequential consistency. This inconsistency is caused by the fact that  $c$  and  $b$  access the same memory location and at least one of them is a store— $c$  is a *conflict operation* with  $b$ . Thus, the two execution orders of  $(b, c)$  and  $(c, b)$  produce different results from each other. Because a conflict operation was executed between  $b$  and  $a$ , the out-of-order execution of  $b$  produces a different result from the in-order execution and it may lead to an inconsistency. To avoid this inconsistency, in previous schemes [3][2][6], P0 nullifies the result of the out-of-order execution of  $b$  when it receives the invalidation request of  $c$ , then re-issues  $b$ .

On the other hand, if the invalidation request of  $c$  is delayed until  $a$  is complete, we can avoid the re-issue of  $b$ . By delaying the invalidation request, we can guarantee that the result of  $b$  is the same as the in-order execution of  $b$  because  $c$  will never be executed between  $b$  and  $a$ —the execution order of  $(b, a, c, d)$  is enforced by delaying the invalidation request of  $c$ . Delaying coherence request does not affect the correctness of a program because operations from different processors can be performed at any order. Delaying the coherence request has been proposed by Adve and Hill [1] to perform synchronization operations out of order without violating ordering constraints.

As seen by above example, the out-of-order execution of an operation does not violate consistency constraints as long as the coherence request to the accessed block is delayed until the operation can be issued in-order. In this paper, we propose the RRB technique, which allows an operation to be committed out of order. Unlike previous schemes [3][6] which relies on speculative commitment and rollback, proposed scheme commits an operation non-speculatively while consistency constraints are guaranteed by delaying coherence requests. Note that an operation is committed out of order only if a load is about to be retired or a store has already been retired but buffered in the store buffer (or memory queue). Thus, the RRB technique does not cause an imprecise exception.

The request reorder buffer (RRB), which is a special buffer between processor and cache controller, takes charge of delaying coherence requests. When a processor commits an operation out of order, the address of the accessed cache block

is stored in the RRB. Whenever a coherence request is received, cache controller should check the RRB before it processes the coherence request. If the target address of the coherence request matches in the RRB, the request is delayed until the prior operations of the committed operation (pointed by a field in the RRB) are complete. Fig. 2 shows an example of out-of-order commitment using the RRB technique.

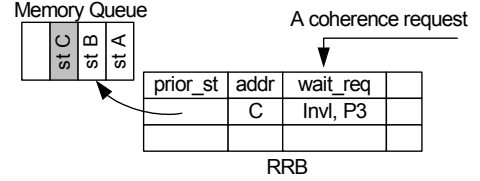


Fig. 2. Example of out-of-order commitment. If store C is committed out of order, the address C is registered in the RRB. When a coherence request to C is received, the request is stored in the RRB until prior stores are complete.

### B. Deadlock Avoidance

Because an operation may be delayed indefinitely, the RRB technique may cause a deadlock. In the sample program in Fig. 1, suppose that P0 commits  $b$  bypassing  $a$  and the invalidation request of  $c$  is delayed at P0. There is a *wait-for dependency* between  $a$  and  $c$ , which is denoted by  $a \rightarrow c$ :  $c$  waits for the completion of  $a$ . In this situation, if P1 also commits  $d$  out of order, the invalidation request of  $a$  should be delayed until  $c$  is complete. Thus, there is also a wait-for dependency  $c \rightarrow a$ . This cyclic dependency leads to a deadlock situation. In this paper, we propose a deadlock avoidance scheme which limits the out-of-order commitment based on the address of the memory block. From now on, we will denote the address of memory block accessed by an operation  $x$  by ' $\&x$ '.

**Deadlock avoidance scheme:** Processors are allowed to commit an operation  $x$  bypassing an operation  $y$ , if and only if  $\&x > \&y$ .

**Correctness:** To perform the proof by a contradiction, suppose that the RRB technique with proposed deadlock avoidance scheme makes cyclic wait-for dependencies among several processors as follows.

$$a_0 \rightarrow a_1 \rightarrow \dots \rightarrow a_n \rightarrow a_0 \quad (a_k: \text{memory operations}) \quad -- (1)$$

A wait-for dependency  $a_k \rightarrow a_{k+1}$  implies that  $a_{k+1}$  accesses the same memory location with an operation  $x$  which is committed bypassing  $a_k$  ( $\&x = \&a_{k+1}$ ). By proposed scheme,  $x$  can bypass  $a_k$  if and only if  $\&a_k < \&x$ , thus,  $\&a_k < \&a_{k+1}$ .

Generally, the supposed dependency (1) means

$$\&a_0 < \&a_1 < \dots < \&a_n < \&a_0$$

It is a contradiction. Therefore, there is no cycle caused by out-of-order commitment. ■

As an example, in Fig. 1, if  $\&b > \&a$ , P0 is allowed to commit  $b$  bypassing  $a$  but P1 cannot commit  $d$  out of program order. Thus, cyclic wait-for dependency is not created and deadlock is avoided.

Although the proposed scheme avoids deadlock, it limits the performance because operations may not be committed out of order due to the deadlock avoidance condition. In general, the more operations are committed out of order, the higher performance is achieved. With proposed deadlock avoidance scheme, the performance is highly dependent on the memory



blocked for resources. Thus, we can avoid deadlock.

### III. PERFORMANCE EVALUATION

We used RSIM [5] to simulate cc-NUMA system with 16 processors. Table I shows the base system configuration. Benchmark applications are from the SPLASH2 suite, except for Mp3d from SPLASH. Table II gives the input sizes used for the benchmark applications. We assume a relatively large L2 cache to eliminate capacity and conflict misses, so that performance difference among the memory models is solely due to the intrinsic behavior of the models.

Table I. Simulated architecture.

SYSTEM PARAMETERS	
CPU	4-issue per cycle
Reorder buffer	64 instructions
Memory queue	64 instructions
L1 cache	16KB, direct-mapped
L2 cache	4MB, 4-way assoc.
L2 fill latency local	41 processor cycles
L2 fill latency remote	117 processor cycles
Cache line size	32 bytes

Table II. Application parameters.

APPLICATION	INPUT PARAMETER
Radix	512K keys
Ocean	128x128 ocean
Barnes	4K particles
Mp3d	50000 particles
Raytrace	Balls4

In our experiments, all implementations use non-blocking caches, hardware prefetch, speculative load execution and store buffering. We set the RRB entry size to 64. We simulated five implementations; SC, TSO, SC+RRB, TSO+RRB, and RC. Note that the performance of TSO is the upper limit of relaxing store-to-load constraint in SC. RC is the upper limit of relaxing all of the ordering constraints.

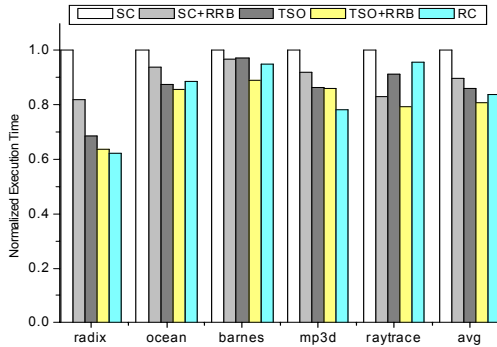


Fig. 4 Normalized execution time relative to SC.

Fig. 4 shows the execution time of benchmarks normalized to SC. In Radix and Raytrace, 17.9% and 16.8% of the execution time were reduced in SC. On the average, the RRB technique achieves the performance improvement of 10.5% in SC and 6.3% in TSO. The performance gap between SC+RRB and TSO is within 3.8%. In Barnes and Raytrace, SC+RRB outperforms TSO because these applications are more sensitive to store-to-store ordering. Comparing to RC, gap between SC+RRB and RC is within 6.4% and TSO+RRB even outperforms RC. It is because the RRB technique effectively handles the contention on a lock variable by committing *unlock* operation out of order and delaying early

fetches to the lock variable. Those early fetches are due to the spinning on a lock variable. If they are not delayed, they usually degrade the performance of lock hand-off.

64 entries of RRB were sufficient. In most applications, the proposed deadlock avoidance scheme was too restrictive and sensitive to memory access patterns of applications.

Fig. 5 shows the normalized execution time when we increase the network latency to 5 times the remote latency described in table 1. On increasing the network latency, the performance gain by the RRB technique increased. In TSO, 12.1% of execution time was reduced by the RRB technique on the average.

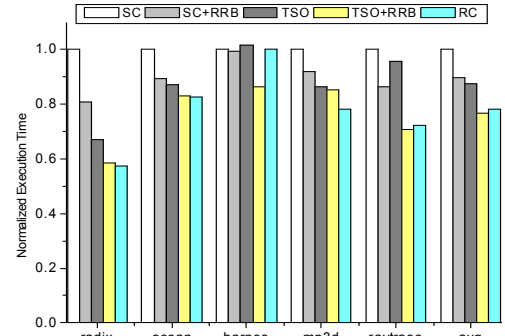


Fig. 5 Impact of network latency.

### IV. CONCLUSION

We have presented the RRB technique to enhance performance of strict consistency models. With the RRB technique, memory operations can be committed out of program order without violating consistency constraints. Current proposal limits the performance gain to avoid deadlock. We expect that deadlock avoidance condition could be more relaxed through exploiting memory access patterns of applications, or relocating address of memory operations by compilers.

### ACKNOWLEDGMENT

This research is supported by KISTEP under the National Research Laboratory program. The authors thank Sihm, KueHwan and SoYeon Park for their insightful discussions and help.

### REFERENCES

- [1] S. V. Adve, M. D. Hill, "A Unified Formalization of Four Shared-Memory Models. IEEE Transactions on Parallel and Distributed Systems 4(6): 613-624, 1993.
- [2] K. Gharachorloo, A. Gupta, and J. Hennessy, "Two Techniques to Enhance the Performance of Memory Consistency Models," Proc. Int. Conf. on Parallel Processing, pages 355-364, August 1991.
- [3] C. Gniady, B. Falsafi, and T. N. Vijaykumar, "Is SC+ILP = RC?," Proc. Int. Symp. on Computer Architecture, pages 162-171, May 1999.
- [4] J. Laudon, and D. Lenoski. "The SGI Origin: A cc-NUMA Highly Scalable Server," Proc. Int. Symp. on Computer Architecture, May 1997.
- [5] V. S. Pai, P. Ranganathan, and S. V. Adve, "RSIM: A Simulator for Shared-Memory Multiprocessor and Uniprocessor Systems that Exploit ILP," Proc. Workshop on Computer Architecture Education, 1997.
- [6] P. Ranganathan, V. S. Pai, H. AbdelShafi, and S. V. Adve, "Using Speculative Retirement and Larger Instruction Windows to Narrow the Performance Gap Between Memory Consistency Models," SPAA, pages 199-210, June 1997.