

A Speculative Coherence Scheme using Decoupling Synchronization for Multiprocessor Systems

Kue-Hwan Sihn, Joonwon Lee, Jung-Wan Cho

Dept. of Computer Science, Korea Advanced Institute of Science and Technology, Korea
{khsihn, joon, jwcho}@calab.kaist.ac.kr

Abstract—This paper proposes a new speculative coherence scheme, SCDS, for hardware distributed shared memory systems to reduce the overhead of coherence action in directory-based cache-coherence protocol. SCDS has two main features, predicting accurate timing of speculative coherence with synchronization information and detecting write pattern (migratory and non-migratory) for exclusive blocks' speculative coherence action. In our simulation, SCDS outperforms existing schemes (DSI and LTP) for well-synchronized applications.

Keywords— distributed shared memory, self-invalidation, speculative coherence

I. INTRODUCTION

DIRECTORY-based cache-coherence protocols are widely used for hardware DSM (distributed shared memory) systems [3], but they suffer from overhead for processing coherence messages. The aim of *speculative coherence* mechanisms is to reduce the overhead of performing such coherence action by speculatively changing the state of cached block in a node. Speculative coherence mechanisms can be either static or dynamic. Several dynamic schemes that can reflect run-time behavior of applications without user intervention have been proposed.

Dynamic self-invalidation (DSI) [4] selects the block which shows the producer-consumer pattern, and then invalidates the selected memory blocks in the cache when the processor reaches at a synchronization point. Because of its lack of timing prediction capability, this scheme blindly invalidates the block earlier than necessary at every synchronization point when the same block is accessed over several synchronization points. Early invalidation induces extra latency for accessing the block again.

Last touch predictor (LTP) [2] was proposed to overcome such handicap. It tracks the sequence of instructions that touch a memory block until a coherence message arrives from the directory, and stores the trace for the block. LTP selects a block which repeats the stored trace as a speculative coherence candidate, and initiates speculative coherence action when the replay of the trace ends. This scheme predicts the accurate timing of speculative coherence action at a cost of watching every memory access behavior from the processor to the cache. It needs special type of integrated processor which includes

DSM controller and the predictor in it. Storing all prediction values in the processor needs large storage on the chip. In addition, we found that LTP performs poorly for irregular access patterns like competing accesses in synchronization constructs or false sharing.

Although ignored in existing schemes, determining the type of speculative coherence action can be another important issue. An exclusively cached block can either be invalidated or downgraded (state changes to "shared") depending on the type of the protocol employed. When invalidation is the only option like in DSI and LTP, unnecessary misses will occur when the previous writer reads the block again.

In this paper, we propose a new speculative coherence mechanism, Speculative Coherence scheme using Decoupling Synchronization (SCDS) which predicts the timing and types of coherence action based on synchronization information. Our scheme does not need to watch every memory access in the processor, and it is shown to be less sensitive to competing accesses and false sharing. We introduce the rationale of our scheme in the next section, and then describe the architecture before exploring performance of three dynamic speculative coherence schemes.

II. BASIC BEHAVIOR OF SCDS

A memory block exhibiting frequent sharing begins its life in the cache¹ with a coherence miss and ends the life with a coherence message for the block. SCDS tracks the lifetime of the cached memory block with PCs (program counter) of two synchronization operations. PC of the first synchronization (PC_1) is used as "index" and it identifies the beginning of lifetime. PC of the second one (PC_D) is used as "prediction value" and it conveys the timing of triggering speculative coherence action. We call the second synchronization *decoupling synchronization* because it is considered to *decouple* conflicting accesses.² PC_1 is the recent synchronization when the coherence miss occurs for the block, and PC_D is updated with the latest synchronization when the coherence message for the block arrives. Once index-prediction pair is created, the processor performs speculative coherence action at PC_D whenever it gets a coherence miss for the same block after PC_1 . If the block is in exclusive mode,

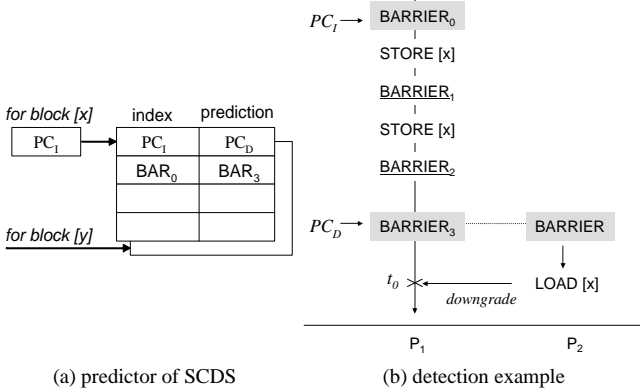
¹ In this paper, *cache* indicates the cache where the sharing of a memory block is managed (L2 cache in our environment).

² Conflicting accesses are accesses from different processors to the same memory block, when at least one processor wants to write [1].

SCDS decides appropriate coherence type for the block.

Using synchronization for the coherence timing is sufficient because a processor usually performs synchronization before and after accessing an actively shared memory block to avoid data race. This approach reduces significantly the amount of information for prediction from the processor by not using detailed instruction history.

A. Detecting Coherence Timing



(a) predictor of SCDS

(b) detection example

Fig. 1. Detecting coherence timing

The predictor of SCDS resides in the directory, and several *prediction records* exist per memory block (Fig. 1(a)). A prediction record has an index-prediction pair and other information, and it corresponds to a block's lifetime in the cache. Index (PC_I) of the prediction record is determined according to the latest synchronization (except UNLOCK) when the block is fetched or ownership for the block is obtained. For example, in Fig. 1(b), if the first STORE [x] of processor 1 (P1) causes a coherence miss and gains ownership, then PC_I will be BARRIER₀. UNLOCK is not used for index because of its semantics. Once index for a prediction record is decided, PC_D should be selected carefully. Since SCDS does not use the last touch information of memory accesses unlike LTP, it is hard to identify the first synchronization following the last touch (BARRIER₂ in Fig. 1(b)). SCDS uses rather conservative approach that selects the last synchronization (BARRIER₃ in Fig. 1(b)) as PC_D when the coherence message for [x] arrives at t_0 . In our simulation, selecting the last synchronization was not too late for most cases. DSI tends to trigger too early (BARRIER₁), thus unnecessary cache misses for the block will arise.

If the index of the prediction record is LOCK, SCDS handles it differently. For blocks protected by LOCK, the possibility of accessing them outside critical section is low. Therefore, SCDS selects the last UNLOCK as PC_D rather than last non-lock synchronization in order to trigger coherence action as early as possible when the index is LOCK.

B. Detecting Coherence Type

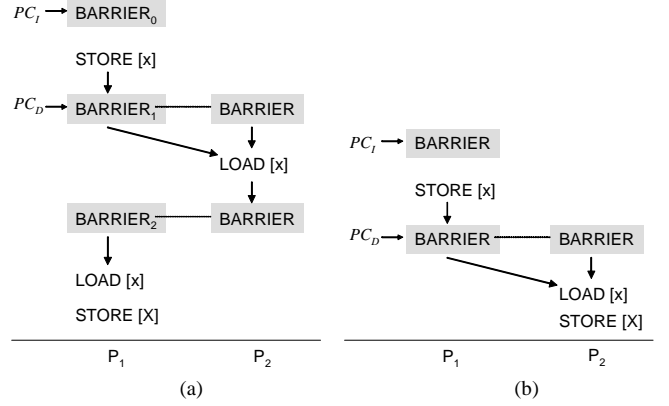


Fig. 2. Two types of write patterns

Appropriate coherence type for exclusively cached block is determined by its write patterns. SCDS detects migratory/non-migratory write patterns based on synchronization. Fig. 2(a) shows a typical non-migratory write pattern while Fig. 2(b) shows migratory one. In Fig. 2(a), to prevent read miss in P1 after BARRIER₂, downgrade should be performed at PC_D . On the other hand, to let P2 get the ownership from the directory without further invalidation in Fig. 2(b), invalidation should be performed at PC_D .

Because it is unknown whether a writing instance is migratory or not when downgrade message arrives at P1, another request should be examined to determine correct write pattern. As seen in Fig. 2(b), if P2 requests the ownership after PC_D of P1 (and before other synchronization in P1), this write case is regarded as migratory. Otherwise, the write case is thought to be non-migratory. This decision is stored in special bit, FI (*force-invalidation*) in the prediction record. This bit indicates that the block should be invalidated rather than downgraded when speculative coherence action is triggered. This information can be used for further migratory optimization [8].

III. THE SCDS ARCHITECTURE

SCDS is composed of prediction table and triggering logic. Prediction table is located at the directory and manages prediction. Prediction process begins with a cache miss in a processor. PC_I value is attached to every request and sent to the prediction table. If the matching prediction record for given PC_I value is found, the prediction table returns PC_D value. This prediction result is sent to the requester for later speculative coherence action. If no matching record is found, the prediction table creates a new prediction record and waits until the cache block gets coherence message from the requester to learn PC_D value. PC_D value is brought to the prediction table by piggybacking on the coherence acknowledgement.

Triggering logic is in charge of tracking synchronization for PC_I and PC_D in a processor and initiating speculative coherence action when a new synchronization starts. After triggering, the prediction table verifies the result of speculative

coherence action.

A. Prediction Table

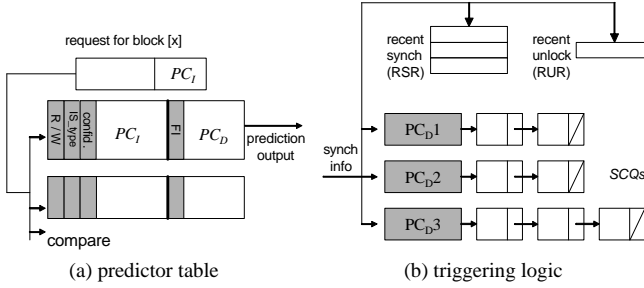


Fig. 3. Block diagram of SCDS

The predictor table resides in the shared memory directory controller, and one memory block may have several prediction records which are shared by all processors. It inspects incoming request packets and predicts PC_D according to PC_I of the request (Fig. 3(a)). In this figure, prediction records for one memory block are shown.

In a prediction record, the index has extra bits besides PC_I; R/W bit is used for discriminating read and write, and index_synch_type bit (IS_type in the figure) denotes synchronization type (LOCK or non-lock). Confidence value (confid. in the figure) is 2-bit saturation counter used in verification process. The prediction result has two subfields (FI, PC_D); FI bit is the result of migratory detection as mentioned earlier. This result is piggybacked on reply packets to the processor.

B. Triggering Speculative Coherence Action

SCDS assumes synchronization detection by special instruction, which can be implemented alternatively by writing to special area of memory or I/O address. This instruction exposes synchronization type (non-lock, LOCK, UNLOCK) and its PC to the triggering hardware. Triggering hardware tracks this synchronization information in the recent synchronization register (RSR) and the recent UNLOCK register (RUR). RSR holds the PC and type of recent synchronization for PC_I of outgoing request, and RUR contains the PC of the recent UNLOCK regardless of its lock variable (Fig. 3(b)). RSR changes when the synchronization type is not UNLOCK, but LOCK and non-lock because the semantics of UNLOCK is not adequate for labeling a new synchronization era. In case of nested critical section, RSR is organized as a stack and it is popped when the processor exits a critical section. When learning for an access is in progress, PC_D is chosen according to the synchronization type of PC_I (piggybacked on the coherence message from the directory) for the currently cached block. If the type of PC_I is non-lock, the value of RSR is used for PC_D. Otherwise PC_D is updated with the value of RUR for the LOCK type of PC_I.

On a successful prediction, a processor receives the prediction result (address of the block, FI, R/W) for the

requested block. The prediction result is stored into a queue indexed by PC_D value. This queue is called Speculative Coherence action Queue (SCQ) in Fig. 3(b). When the processor reaches a synchronization point, triggering mechanism compares the PC of current synchronization with PC_D values from SCQs' index. If a match is found, then entries from the matching queue are dequeued to initiate speculative coherence action. In actual speculation process, read-only cache blocks with R/W bit 'read' are invalidated while actions on exclusive cache blocks with R/W bit 'write' depend on the FI bit. If the FI bit is set, the block is invalidated. Otherwise, the block is downgraded since the block is predicted to be read again by the processor. Then the acknowledgement is sent to the prediction table in the directory.

C. Verification of Prediction Result

To verify the result of prediction, SCDS uses verification mask similar to that of LTP. Verification mask is an array that remembers the processors which committed speculative coherence. If a processor committed speculative coherence earlier than necessary, this can be easily found via the verification mask. On such premature speculation, the prediction record associated with the commitment is marked as *early* by decrementing the 2-bit saturation counter (*confid.* in Fig 3(a)) in the record. If the coherence action is correct, the confidence counter will be increased. Indices to the prediction records should be remembered for this use.

IV. PERFORMANCE EVALUATION

We use RSIM [5] to simulate a mesh-network connected, 16-processor hardware DSM system. The base system parameters are given in Table 1. In benchmark applications, *ocean*, *water*, and *barnes* are from the SPLASH-2 [9] suite, *mp3d* is from SPLASH [7], and *tomcatv* is from Wind Tunnel [6]. Parameters for benchmark programs are given in Table 2. We compared DSI, LTP, and SCDS with the base performance.

Table 1. Simulated architecture

SYSTEM PARAMETERS	
CPU	Max. 4-issue / cycle
Number of processors	16
Instruction window	64 instructions
Cache block size	32 bytes
L1 cache	16KB, writeback
L2 cache	1024KB, off-chip
L1/ L2 hit latency	1 cycle/ 8 cycles
Memory access time	18 cycles
Network speed	1/3 of processor speed
Network width	64 bit
Flit delay	6 cycles

Table 2. Application parameters

BENCHMARK	DATA SETS
<i>Ocean</i>	128 × 128
<i>Water</i>	343 molecules, 4 steps
<i>Tomcatv</i>	128 × 128, 50 iterations
<i>Barnes</i>	4096 particles, 3 iterations
<i>mp3d</i>	50000 molecules, 10 steps

Sequential Consistency is used to see the maximum benefit from the speculative coherence in our simulation. The performance of speculative coherence would be different for more relaxed consistency, and it needs further investigation.

In our simulation, the performance of LTP is affected by competing accesses and false sharing. In LTP, a history of *competing* accesses [1] to a memory block changes variously. Competing accesses are usually found in synchronization constructs such as spinning lock or user-programmed spinning routines. A number of processors involved in the competing accesses contend for a memory block simultaneously, which makes traces different every time and hence the increased possibility of miss prediction. Similarly, in the presence of false sharing, different history patterns will appear. As this situation repeats, the history will be garbled, resulting in low performance. DSI and SCDS are less sensitive to these effects because their timing is based on the synchronization. However, triggering at synchronization makes a bursty traffic and may limit the performance.

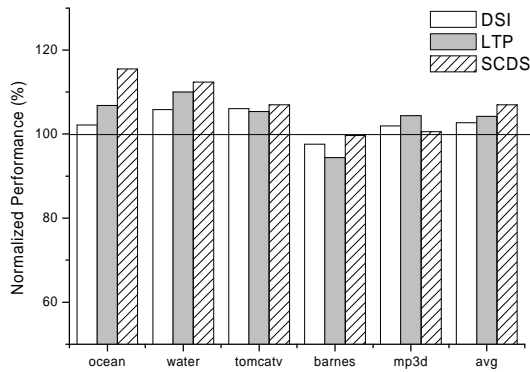


Fig. 4. Normalized performance

Vertical axis in Fig. 4 denotes the performance normalized to the base architecture (100%). Average speedup of SCDS (7.0%) is better than the other two existing schemes (4.2% and 2.7% for LTP and DSI, respectively). SCDS mainly benefits from write pattern detection and avoidance of contended accesses and false sharing. DSI shows the low coverage and early invalidation problems for *ocean* and *water*.

In *ocean*, SCDS achieves 15.5% of speedup largely due to its appropriate action for non-migratory write pattern. In *water* and *tomcatv*, LTP suffers from contended accesses, while SCDS and DSI do not. The performance of LTP in *barnes* is eroded by much of false sharing and contended accesses. Because SCDS is more robust to false sharing than LTP, its slowdown is not serious. *Mp3d* has data races, therefore SCDS fails to find decoupling synchronization for many access patterns. DSI is less affected in *mp3d* because it invalidates blocks at the nearest synchronization despite lack of decoupling synchronization.

The effect of false sharing increases for larger memory block size. When the memory block size is 128 byte, LTP's average speedup is -2.7% (graph is not presented), actually

increasing the execution time. On the other hand, speedups of SCDS and DSI with 128-byte memory block are 2.2% and 1.1%, respectively.

V. CONCLUSION

In this paper, we proposed a new speculative coherence scheme, SCDS, and attained some knowledge about the actual speculative coherence mechanisms. First, we showed it is possible to predict self-coherence timing without tracking every memory access from the processor. Although SCDS needs explicit synchronization, isolation of prediction logic from the processor eases the deployment of SCDS for a wide spectrum of processors. Second, synchronization construct and false sharing may affect speculative coherence scheme if the scheme does not consider synchronization. Among the evaluated schemes, LTP was the most sensitive to the contended accesses and false sharing. Third, if migratory-favor protocol is not used, it is useful to differentiate write patterns, e.g. migratory pattern from non-migratory pattern. Conventional migratory optimization [8] can help LTP and DSI, but it lacks prediction capability. In contrast, SCDS predicts the write pattern for each cache miss appeared in an execution.

ACKNOWLEDGMENT

This research is supported by KISTEP under the National Research Laboratory program.

REFERENCES

- [1] K. Gharachorloo, S. V. Adve, A. Gupta, J. L. Hennessy, and M. D. Hill, "Programming for different memory consistency models," *Journal of Parallel and Distributed Computing*, 15(4):399-407, Aug. 1992.
- [2] A-C. Lai, and B. Falsafi, "Selective, accurate, and timely self-invalidation using last-touch prediction," *Proc. 27th Ann. Int. Symp. on Computer Architecture*, pages 139-148, Jun. 2000.
- [3] J. Laudon, and D. Lenoski. "The SGI Origin: A cc-NUMA highly scalable server," *Proc. 24th Ann. Int. Symp. on Computer Architecture*, pages 241-251, May 1997.
- [4] A. R. Lebeck, and D. A. Wood, "Dynamic Self-Invalidation: Reducing coherence overhead in shared-memory multiprocessors," *Proc. 22nd Ann. Int. Symp. on Computer Architecture*, pages 48-59, Jun. 1995.
- [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] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood, "The Wisconsin Wind Tunnel: Virtual prototyping of parallel computers," *Measurement and Modeling of Computer Systems*, pages 48-60, 1993.
- [7] J. P. Singh, W. D. Weber, and A. Gupta, "SPLASH: Stanford parallel applications for shared-memory," *Computer Architecture News*, vol. 20, pages 5-44, March 1992.
- [8] P. Stenström, M. Brorsson, and L. Sandberg, "An adaptive cache coherence protocol optimized for migratory sharing," *Proc. 20th Ann. Int. Symp. on Computer Architecture*, pages 108-118, May 1993.
- [9] S. C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," *Proc. 22nd Ann. Int. Symp. on Computer Architecture*, pages 24-36, July 1995.