Using Coherent Value Speculation to Improve Multiprocessor Performance

Jichuan Chang[†], Jaehyuk Huh [§], Rajagopalan Desikan [§], Doug Burger [§], Gurindar Sohi[†]

[†]Computer Sciences Department
University of Wisconsin-Madison

[§]Dept. of Computer Sciences
The University of Texas at Austin

Abstract

Transmission of cache lines in cache-coherent shared memory machines is necessary for communication but can cause significant latencies across the system. The ongoing growth in cache capacities shifts the distribution of cache misses from capacity and conflict misses to coherence misses, which consist of misses caused by both true and false sharing. In this paper we propose coherence decoupling and coherent value speculation to improve multiprocessor performance. ***Coherence decoupling*** *splits the caching operation from the coherence operation, allowing processors to speculate on the load value or the data access permission, despite not having both for that operation.* ***Coherent value speculation*** *provides speculatively-coherent values for a coherence-decoupled system. In this paper we focus on using shared value prediction for coherent value speculation. We describe and profile the potential of three value prediction schemes:* address-cache-based*,* PC-table-based*, and* PC+address-table-based*. The first scheme simply uses the stale values existing in the cache, while the other two predict using recent value tables. We present early results that suggest excellent potential for coherent value speculation.*

1 Introduction

Communication due to coherence operations is becoming increasingly expensive in cache-coherent shared-memory machines. As system caches grow in size and/or associativity, coherence misses (read accesses to invalid data and write accesses to shared or invalid data) become an increasingly large problem as capacity and conflict misses diminish.

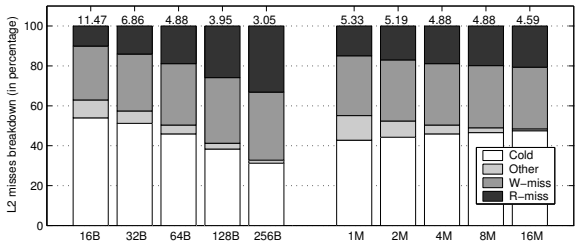

Figure 1: **L2 misses breakdown for an OLTP benchmark** *(On top of each bar shows the number of L2 misses per thousand instructions. On the left-hand side, the cache line size is varied from 16 bytes to 256 bytes, while the number of sets is fixed; on the right, the cache line size is fixed to 64 bytes, while the cache size is increased from 1MB to 16MB.)*

To demonstrate that trend, we collected traces from an on-line transaction processing (OLTP) benchmark, and partition the L2 cache misses into (1) Cold-start misses, (2) Read misses on shared data that have previously been invalidated (or R-misses, for "Read" misses on shared data), (3) Write misses causing invalidations (or W-misses, for "Write" misses on shared data), and (4) Other misses (capacity/conflict misses). Both R-misses and W-misses are coherence misses. Figure 1 shows that coherence misses account for half of the total L2 misses with 64-byte lines for large caches. As the line size is increased, the number of total misses decreases, but the number of R-misses increases due to false sharing. In the 256-byte line system, which displays the smallest total number of misses, W-misses diminish much more slowly than non-coherence misses, making the coherence misses account for over two-thirds of the total L2 misses. Coherence misses thus have the potential to be a major limiter of performance in future systems.

The essence of data communication in a shared memory machine is to observe the right value at the right time, where correctness is defined by given memory consistency model [1]. Previous research has focused on relaxing the orders among memory operations to allow overlapping among them, while keeping the programming interface simple and intuitive. The approach described in this paper is somewhat different: we break each coherence operation into distinct steps and allow some of those steps to proceed independently. This *coherence decoupling* permits the execution of a memory operation to proceed speculatively, in parallel with the coherence request for permission to complete the operation.

Such a decoupled approach, combined with speculative operations on shared data, exposes a new set of strategies to attack the problem of coherence misses, some of which we explore in the rest of this paper. In Section 2 we first discuss what steps are needed to serve a miss to shared data, and how they can be speculatively decoupled and executed in parallel. We also study three schemes for coherent value speculation: *address-cache-based*, *PC-table-based*, and *PC+address-table-based*. Preliminary results on the potential of these techniques are presented in Sections 3. Section 4 briefly describes related work, and Section 5 concludes with our planned future work.

2 Coherent Value Speculation

To service a shared miss, both the data value and proper access permission to the data must be obtained. Conventional cache-coherent implementations provide both the value and the (implicit) permission in a single reply. Requesting a remote coherence permission for a load or store takes tens or hundreds of cycles to finish, typically causing the requesting processor to stall.

However, the correct data item may reside in other caches in the system, including the requestor's cache, despite the lack of permission to read from the line. Moreover, values in these locations are natural candidates for value prediction, even though the corresponding cache lines may not contain the most recent value. Speculatively using a predicted value and verifying the result later could hide much or all of the miss latency.

Similarly, writes to shared data can also be broken into steps and speculatively executed. One application of this mechanism is to enhance the existing optimization of read-modify-write sequences [14]. Such a sequence consists of ordered read, modify and write operations on the same data, so issuing a get-exclusive request when it is first loaded will save the upgrade operation upon write. Separately obtaining the value and permission allows further optimization, in which the value part of a get-exclusive request is served in parallel with the actions required to invalidate other sharers.

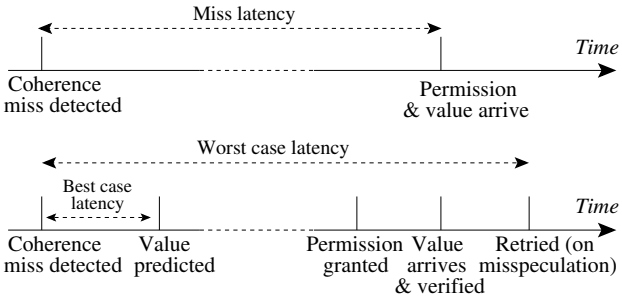

Figure 2: **Coherent Value Speculation**

Figure 2 provides an example of coherence decoupling using coherent value speculation. In this scheme, the value of shared data is locally predicted upon a coherence miss, and immediately used by the processor. The real coherence request is sent out in parallel, and eventually completes with the proper value and permission. Here the value and permission can come back in either order, as long as atomicity appears to be maintained between the first use of the value and the commitment of the load instruction. Martin et al. discussed how to detect consistency violations in this process for correctness reasons [11]. Finally the value is verified. If the prediction is correct and no violation is detected, the speculation succeeds; otherwise it results in a misspeculation, the dependent instructions should be squashed, and the load retried. During this period, dependent results are buffered locally to allow rollback in case of misspeculation. The chance of misspeculation can be reduced by better predictor or selective speculation, and the misspeculation penalty can be mitigated by selective re-execution.

The figure also compares the speculative execution with sequential execution in terms of latency. The top half of the figure depicts the events in a sequential execution and its miss latency. The bottom half shows the events in a speculative execution. If the prediction is accurate, value

speculation on shared data experiences only the best case latency, which is much less than the latency of sequential execution. The key to higher performance in this case is better value prediction accuracy and coverage.

There are many approaches to get speculative values, which can be classified by (1) where the values are stored, and (2) what information is used to search for the values. In the following subsections, we study three schemes that implement coherent value speculation. (1) address-cache-based speculation, in which the data cache is used as the storage for values, (2) PC-table-based and (3) PC+address-table-based speculation, in which PC and PC+address, respectively, are used to index into separate recent value tables.

Address-cache-based: This scheme exploits the "stale" value stored in the cache upon a shared load miss, speculating that either the miss is caused by false sharing, or the silent stores [8] and temporally silent stores [9] have left the value unchanged. This scheme requires less area by using the local L2 cache as the prediction table, while focusing on an important subset of shared value speculations. Section 3.1 will present the accuracies when loading invalid data from the cache for the different types of operations, and different cache sizes.

PC- and PC+address-table-based: In these schemes, value prediction tables are searched upon shared load misses. Either the PC of the faulting instruction or the PC and the data address can be used to index into the recent value table. Conventional load value prediction [10] only uses PC as the index because the data address may not be available when the prediction is made. However for coherence misses, not only is the address already known, but the long miss latency allows for slower but more accurate value predictors. Section 3.2 will give some initial data on shared value locality and predictor performance.

3 Evaluation Results

We currently use different multiprocessor simulation infrastructures for cache- and table-based schemes, so we present the results separately.

3.1 Address-cache-based Scheme

In this section we briefly describe our profiling results showing the potential accuracy of exploiting the "stale" values in the cache.

3.1.1 Methodology

We ran six benchmarks – selected from various scientific application suites – on a version of SimOS-PPC that uses IBM's AIX 4.3.1 as the simulated OS. The experiments were performed on a fully execution-driven multiprocessor timing simulator based on the SimpleScalar processor model. Our simulated system models an 8-node SMP multiprocessor system. Each node in the SMP system has 64KB split L1 instruction and data caches, and a unified L2 cache with the size of 1MB, 2MB or 4MB. All caches have 64-byte lines. These caches are kept coherent with a snoopy bus using a MOESI coherence protocol.

Our scientific applications consist of two common types of parallelization models, MPI (Message Passing Interface) and shared-memory. The MPI-based applications are SMG2000 and Sweep3D from the ASCI benchmarks, and LU from the NAS benchmark suite. We used MPICH 1.2.5 as our MPI implementation. The shared-memory benchmarks are Barnes from the SPLASH-2 benchmark suite, and sPPM and MDCASK from the ASCI benchmarks. A standard pthread library is used for synchronization of these shared-memory benchmarks.

3.1.2 Speculation Accuracies

To measure the accuracy of speculation, we measured the number of references that continue to have valid data after invalidations. On a L2 miss, if the cache block is resident in the L2 cache in an invalid state (R-miss), we use the value as a predicted value, and then verify it with the correct value, when it is returned from memory or other L2 caches. Table 1 shows the accuracy of the address-cache-based speculation for all L2 misses incurred, partitioned into regular accesses (correct pred normal) and synchronizing load locked instructions (correct pred synch). For all the benchmarks except sPPM, the address-cache-based speculation correctly predicted the values for more than 50% of R-misses, with a lower accuracy of 30% for sPPM. R-misses may vary from 2% to more than 90% of all L2 misses, and the performance of the applications with high-coherence misses such as sPPM and MDCASK, can be improved significantly by this simple prediction. The results suggest that this scheme has the potential to greatly reduce the miss overhead induced by false sharing, thus allowing for larger L2 cache lines.

Benchmarks	1 MB L2				2 MB L2				4 MB L2			
	# misses	# R-misses	correct pred normal	correct pred synch	# misses	# R-misses	correct pred normal	correct pred synch	# misses	# R-misses	correct pred normal	correct pred synch
SMG2000	10.2M	3.4M	1.8M	0.16M	8.2M	3.5M	1.8M	0.16M	7.1M	3.5M	1.8M	0.16M
Sweep3D	5.8M	0.51M	0.26M	0.06M	3.3M	0.54M	0.25M	0.06M	1.6M	0.55M	0.24M	0.06M
LU	5.0M	0.13M	0.085M	0.018M	3.9M	0.12M	0.067M	0.013M	2.3M	0.13M	0.072M	0.014M
Barnes	0.85M	0.07M	0.05M	0.009M	0.65M	0.11M	0.08M	0.010M	0.50M	0.14M	0.11M	0.011M
sPPM	36.6M	31.2M	11.7M	0.03M	34.8M	30.8M	11.7M	0.03M	34.6M	31.0M	11.6M	0.04M
MDCASK	11.0M	10.3M	7.7M	0.004M	10.2M	9.6M	7.1M	0.004M	10.8M	10.2M	7.7M	0.004M

Table 1: Speculation Accuracy for 1MB, 2MB, and 4MB L2 Caches

3.2 Table-based Schemes

3.2.1 Methodology

We use five commercial and scientific benchmarks from the Wisconsin commercial workload suite [2] to examine shared load value locality. These workloads include OLTP (TPC-C), Apache (static web content serving), JBB (a Java server benchmark), plus Barnes-Hut (16K bodies) and Ocean (514x514 grid) from the SPLASH-2 benchmark suite. A 16-node system similar to Sun E10000 is modeled, each node with split L1 instruction and data caches (each is 128 KB, 4-way set associative, using 64-byte blocks), a unified L2 cache (4 MB, 4-way set associative, 64-byte blocks). An MOSI snooping protocol keeps the caches coherent. To simplify the analysis, no store buffer is used.

Memory access traces (whose lengths vary from 1.7 to 3.0 billion instructions) are obtained to identify shared load misses. Load-modify-store style atomic accesses are included in traces, but separated from normal (or non-atomic) loads in our results.

	OLTP	Apache2	JBB	Barnes	Ocean
# L2 Miss	11.6M	10.1M	5.0M	3.1M	0.55M
# Shared-LD	3.14M	4.58M	1.43M	1.27M	0.21M
(%/L2-Miss)	27.1%	45.2%	28.5%	40.6%	38.9%
# Normal-LD	2.43M	3.48M	1.16M	9.06M	0.16M
(%/Shared-LD)	77.5%	76.0%	80.7%	71.3%	76.1%

Table 2: Benchmark Statistics

Table 2 lists the statistics of the benchmarks, on the number of L2 cache misses, shared load misses, and shared load misses by non-atomic accesses. It also shows that 27% to 45% of L2 cache misses are R-misses, and 70% to 80% of the shared load misses are made by normal non-atomic load instructions.

3.2.2 Potential Value Prediction Performance

To probe the potential value prediction performance, we feed the traces into counters that collect prediction results for two different prediction schemes. The first scheme is per-PC last value prediction, in which the last value seen by the given PC on a given processor is predicted as the outcome of the miss. The predictor is probed upon every R-miss, updated with the actual load value, and the predictor statistics are updated according to whether the prediction is correct or not.

The other scheme is per-PC-address last value prediction, which exploits the data address information available upon a miss. In this scheme, a last value cache with an infinite number of sets and 16 ways within a set is used. The cache is indexed by the PC, tagged by data block address, and maintained by LRU replacement. No prediction is made if it misses in the table, otherwise the cache is updated with actual load value, and per-PC statistics is updated accordingly. At maximum, 16 addresses are cached for a given PC to save table space and replacement time. This serves as a limit study because an infinite number of sets (PCs) are supported by the value table. This scheme gives better results on many PCs seeing different but predictable values from different addresses.

We observe that not only do a small number of PCs contribute to the majority of correct predictions, but also a small number of (probably different) PCs cause the majority of mispredictions. To filter predictions for unpredictable PCs, we compute the benefit gained by value pre-

	Coverage No-atomic Good PC	Coverage All access Good PC	Coverage All access All PC	Mispred. Ratio Good PC	Mispred. Ratio All PC
PC+addr	14%	29%	62%	0.99%	25%
PC-only	12%	27%	55%	15.86%	44%

Table 3: PC-only vs. PC+address Last Value Prediction

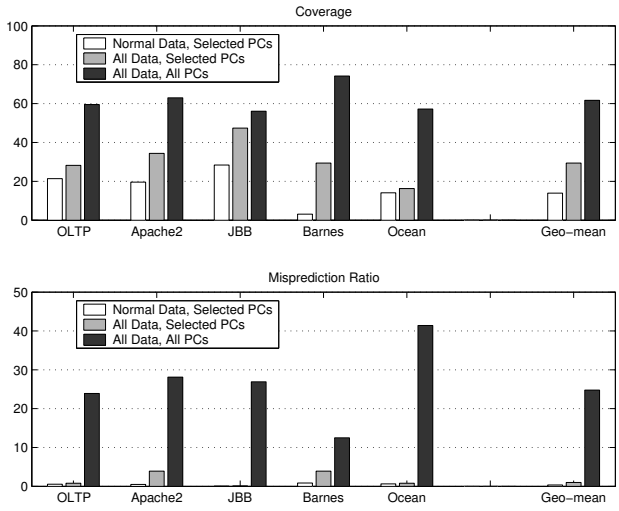

Figure 3: **Percentage of Correct Value Predictions over Total R-misses**

diction on a PC using a simple cost formula (with constants `gain` equals to 4 cycles, and `lose` to 40 cycles):

```
benefit = gain*#correct_pred - lose*#mis_pred
```

Using this formula, the benefits for each PC using different strategies are computed. We choose to use the result of the better strategy (with larger benefit) to mimic the behavior of a hybrid predictor with both PC-address and PC-only predictions. Figure 3 gives the correct prediction coverage for three counting strategies: (1) non-atomic accesses, "good" PCs (where only PCs with positive value prediction benefits are counted); (2) all accesses, "good" PCs; and (3) all accesses, all PCs. On average, 14%, 29% and 62% of all shared load misses are covered by the three strategies, respectively. On the other hand, table 3 compares the prediction accuracy and R-miss coverage of PC-only and PC+address based schemes, it shows that using both PC and address information gives higher coverage and much lower misprediction ratio.

Although considering all accesses on all PCs renders the best coverage, its misprediction ratio is high, 24.8% on average. On the contrary, the average misprediction ratio for "good" PCs is only 0.99%. This suggests that adding confidence levels to value predictor is beneficial.

Figure 4 shows that a small fraction of PCs can cover the majority of correct predictions. On average, using the best 16, 64 and 256 PCs covers 69%, 89% and 97.5% of total correct predictions made on all "good" PCs. This high accuracy shows that small tables can provide most of the benefit of infinitely large last value tables. The PC locality of mispredictions suggests not only that PC-based confidence predictors would work well, but also that future study can focus more on the small number of PCs that produce unpredictable values.

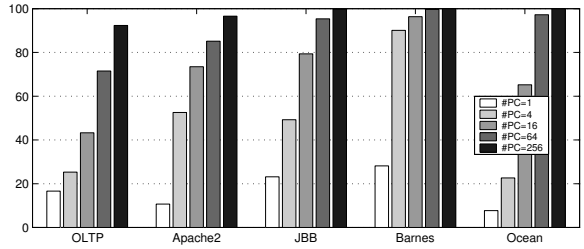

Figure 4: **Percentage of Correct Predictions Covered by Varied Number of PCs**

4 Related Work

The three most pertinent classes of related work include *load value speculation*, *coherence prediction*, and *synchronization speculation*. Coherence decoupling extends previous work on load value speculation to the multiprocessor coherence domain, since previous work on load value speculation has focused on single-threaded codes [10, 16, 17, 3, 4]. In [11], the authors examine the correctness issue when extending value prediction to multithreaded programs or multiprocessors, while coherent value speculation focusing on the performance aspect. The techniques covered in [11] can be used by this work to ensure correct speculation.

Coherence prediction has been proposed in multiple forms by numerous researchers [6, 5, 13, 7]. In it, co-

herence operations are speculatively initiated, predicting the sharing patterns in advance of the coherence requests. Coherence decoupling differs from coherence prediction, since in coherence decoupling the memory operation itself is speculative, while the coherence operation could be either speculative or non-speculative.

The prior work most similar to coherence decoupling is synchronization speculation, in which a lock is speculated to be unheld, permitting speculative entry into critical sections [12, 14, 15]. The similarity consists of speculative access to shared variables, which are limited to locks in synchronization speculation.

5 Conclusions

Coherence decoupling, the breaking of a coherent memory operation into the memory portion and the coherence portion, offers new possibilities for reducing the overhead of sharing in a multiprocessor. In this paper, we have described three schemes for *coherent value speculation*, one application of coherence decoupling, in which either a value prediction table produces a result for a load before the coherence state is known, or the value loaded from an invalidated line is used while the coherence upgrade occurs in the background. The latter succeeds when a load is issued to a falsely shared word in a cache line, or when the remote producer issues a silent store. PC and data address are used to index into the table or the cache. We have shown that these schemes show sufficient accuracy to greatly reduce the performance degradations caused by long-latency coherence operations.

Our future work will first analyze the variation of these policies across our two multiprocessor simulation environments. Future enhancements to these policies will search for more accurate but slower value predictors, exploiting the long latencies of remote multiprocessor coherence operations for more accurate prediction. We will also work to detect consistency violations in our predictors, or make predictions only for loads that do not produce such violations, since recent work [11] has shown that simply verifying that a value eventually read is the same as the value predicted is sometimes insufficient to conform to a defined memory consistency model. Finally, since coherence decoupling is a more general form of synchronization speculation, we plan to study a unified approach for both speculation on normal loads and speculative lock elision [15].

References

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, pages 66–76, 1996.

[2] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, D. J. Sorin, M. D. Hill, and D. A. Wood. Simulating a $2m commercial server on a $2k pc. *IEEE Computer*, 36(2):50–57, Feb. 2003.

[3] B. Calder, P. Feller, and A. Eustace. Value profiling and optimization. *Journal of Instruction Level Parallelism*, 1:1–6, 1999.

[4] B. Calder and G. Reinman. A comparative survey of load speculation architectures. *Journal of Instruction-Level Parallelism*, 2000.

[5] S. Kaxiras and J. R. Goodman. Improving CC-NUMA performance using instruction-based prediction. In *Proc. of the 5th International Symposium on High Performance Computer Architecture*, pages 161 – 170, Jan 1999.

[6] S. Kaxiras and C. Young. Coherence communication prediction in shared-memory multiprocessors. In *Proc. of the 6th International Symposium High Performance Computer Architecture*, pages 156–167, 2000.

[7] A.-C. Lai and B. Falsafi. Memory sharing predictor: The key to a speculative coherent DSM. In *Proceedings of the 26th Annual Int'l Symp. on Computer Architecture (ISCA'99)*, pages 172 – 183, May 1999.

[8] K. M. Lepak and M. H. Lipasti. Silent stores for free. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2000.

[9] K. M. Lepak and M. H. Lipasti. Temporally silent stores. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[10] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. Value locality and load value prediction. In *Architectural Support for Programming Languages and Operating Systems*, pages 138–147, 1996.

[11] M. M. K. Martin, D. J. Sorin, H. W. Cain, M. D. Hill, and M. H. Lipasti. Correctly implementing value prediction in microprocessors that support multithreading or multiprocessing. In *34th International Symposium on Microarchitecture (MICRO)*, December 2001.

[12] J. F. Martínez and J. Torrellas. Speculative synchronization: Applying thread-level speculation to explicitly parallel applications. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 18–29, San Jose, CA, Oct. 2002.

[13] S. S. Mukherjee and M. D. Hill. Using prediction to accelerate coherence protocols. In *Proc. of the 25th Annual Int'l Symp. on Computer Architecture (ISCA'98)*, pages 179 – 190, June 1998.

[14] R. Rajwar and J. R. Goodman. Transactional lock-free execution of lock-based programs. In *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[15] R. Rajwar and J. R. Goodman. Speculative lock elision: Enabling highly concurrent multithreaded execution. In *34th International Symposium on Microarchitecture*, December, 2001.

[16] Y. Sazeides and J. E. Smith. The predictability of data values. In *Proceedings of the 30th International Symposium on Microarchitecture, MICRO-30*, pages 248–258, Dec 1997.

[17] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. In *International Symposium on Microarchitecture*, pages 281–290, Dec 1997.