Memory Coherence Activity Prediction in Commercial Workloads

Stephen Somogyi, Thomas F. Wenisch, Nikolaos Hardavellas,
Jangwoo Kim, Anastassia Ailamaki, Babak Falsafi
Computer Architecture Laboratory (CALCM)
Carnegie Mellon University, Pittsburgh, PA 15213
http://www.ece.cmu.edu/~puma2/

Abstract. Recent research indicates that prediction-based coherence optimizations offer substantial performance improvements for scientific applications in distributed shared memory multiprocessors. Important commercial applications also show sensitivity to coherence latency, which will become more acute in the future as technology scales. Therefore it is important to investigate prediction of memory coherence activity in the context of commercial workloads.

This paper studies a trace-based Downgrade Predictor (DGP) for predicting last stores to shared cache blocks, and a pattern-based Consumer Set Predictor (CSP) for predicting subsequent readers. We evaluate this class of predictors for the first time on commercial applications and demonstrate that our DGP correctly predicts 47%-76% of last stores. Memory sharing patterns in commercial workloads are inherently non-repetitive; hence CSP cannot attain high coverage. We perform an opportunity study of a DGP enhanced through competitive underlying predictors, and in commercial and scientific applications, demonstrate potential to increase coverage up to 14%.

1 Introduction

Modern distributed shared memory (DSM) machines face an ever-growing disparity between processor cycle times and interconnect latencies. Semiconductor fabrication advances and circuit innovations have led to dramatic increases in operating frequencies, and recent architectural developments—such as chip multiprocessors and simultaneous multithreading—place an even greater load on memory subsystems. DSMs face all the challenges associated with uniprocessor designs, as well as coherence requirements that tax the interconnect.

Distributed shared memory is an attractive multiprocessor architecture, capable of scaling to a large number of nodes. Unlike a cluster of independent machines, DSM maintains the familiar programming model of uniprocessors and symmetric multiprocessors, which allows applications to run on a DSM without modification. However, DSM requires mechanisms to ensure memory coherence and consistency, which create interconnect traffic and add latency to critical operations.

Bandwidth limitations in DSM can be overcome by additional communication channels (feasible but costly). Unfortunately, interconnect latency cannot be so easily reduced. Although microarchitectural and memory coherence optimizations such as out-of-order execution and relaxed memory models can hide some of the latency asso-

ciated with coherence traffic [1], they cannot completely overlap the shared read miss latency.

This paper studies two prediction mechanisms that are used to reduce coherence latency in DSM. We derive these predictors from prior work [10,11] in which they were evaluated using scientific workloads. The 2-level Trace-based DownGrade Predictor (2-TDGP) identifies the final store to a cache block prior to a subsequent read by another node and self-downgrades the block, thus eliminating one network hop from coherent read requests by other nodes. The 2-level Pattern-based Consumer Set Predictor (2-PCSP) predicts which nodes will subsequently read (consume) a value that has been written (produced) by another node. Although outside the scope of this paper, such a prediction could be used to forward blocks to consumers, thus obviating the need for a coherent read request at all and reducing effective latency by several orders of magnitude.

Evaluation of architectural proposals continually becomes more sophisticated. Recently, commercial applications have become very important to the research community [2]. Coherence latency has been shown to be a first-order determinant of database performance in multiprocessor systems, and coherence traffic is increasing with the aggregate caching in the memory hierarchy [3]. Thus, we expect commercial applications, in particular online transaction processing (OLTP), to benefit greatly from techniques that optimize coherence activity. It is essential to evaluate the utility of new and existing proposals on this class of application.

Using instruction traces from full-system simulation [15] of shared-memory multiprocessors running scientific and commercial workloads on stock operating systems, we demonstrate:

- **2-TDGP on Commercial Workloads:** We evaluate a 2-level trace-based downgrade predictor for the first time on commercial workloads. We show that, despite long and complex execution paths, 2-TDGP correctly identifies 47%-76% of productions with 13%-22% mispredictions.
- **2-PCSP on Commercial Workloads:** We evaluate a 2-level pattern-based consumer set predictor for the first time on commercial workloads, and show that it cannot predict a significant fraction of consumers. Due to data- and timing-dependent behavior, the sharing patterns in commercial applications are inherently non-repetitive and therefore unpredictable.
- **Competitive Predictor Opportunity:** We observe that 2-TDGP is sensitive to the inclusion of address information in prediction signatures, and the optimum design point varies within and across commercial and scientific applications. We present an opportunity study for a competitive predictor that dynamically varies signature encoding, exposing up to 14% additional opportunity for 2-TDGP.

The rest of this paper is organized as follows. In Section 2 we review related work in the field. Section 3 presents the design of 2-TDGP and 2-PCSP. Section 4 describes our experimental infrastructure and procedures. Section 5 presents the results. We conclude the paper in Section 6.

2 Related Work

Speculative release of shared data from processor caches in a multiprocessor system was first proposed by Lebeck and Wood [13]. Their technique, *Dynamic Self-Invalidation*, triggered invalidation of shared data blocks, identified via coherence protocol hints, at annotated critical section boundaries. *Last Touch Prediction* (LTP), proposed in [11], instead associates invalidation events with the sequence of instructions accessing a cache block prior to its invalidation. By storing PC traces which repetitively lead to invalidation, LTP can trigger self-invalidation immediately upon the last access without the aid of program annotation. Our 2-TDGP is a derivation from LTP that only predicts the release of dirty shared data (downgrades rather than invalidations). Techniques that relax memory order have been shown to effectively hide coherent write latency [1], obviating the need to predict invalidations. However, these techniques cannot fully overlap coherent read latency; therefore, the retrieval (via a downgrade operation) of a value modified by another node remains on the processor's critical path. [11] evaluated LTP using scientific workloads. This paper presents the first evaluation of this class of predictor for commercial applications. In the context of a uniprocessor system, Hu et al. [7] investigated timekeeping approaches for predicting memory system events. These are straightforward to adapt for coherence prediction in DSM, and in this paper we compare 2-TDGP against a *dead-time*-based predictor.

Predicting the subsequent sharers of a newly produced value, which we generically call consumer set prediction (CSP), was first attempted by Mukherjee and Hill in [16]. Their scheme adapted two-level branch prediction [21] to predict coherence messages based on the history of previously received messages. *Memory Sharing Prediction* (MSP) [10] improves upon this approach by eliminating prediction of acknowledgement messages. Further, MSP summarizes the set of consumers for a value without regard to the order each consumer requests the value, enabling the predictor to tolerate reordering of read requests. Our 2-PCSP further optimizes MSP to predict only read requests. As with 2-TDGP, coherent write latency can be addressed with memory ordering optimizations and write messages need not be predicted. Kaxiras presents a taxonomy for the design space of consumer set prediction [8], and classifies CSPs based on their access and prediction functions. In the Kaxiras taxonomy, 2-PCSP is best categorized as address-based access with two-level prediction. However, 2-PCSP differs in that it uses per-node saturating confidence counters in the second-level table to reduce mispredictions. All previous work on consumer set prediction has evaluated predictors using scientific applications; this paper presents the effectiveness of 2-PCSP for commercial workloads.

3 Design

In this section we present the details of our predictors. Section 3.1 presents 2-TDGP, its derivation from previous predictors, and an example of its operation. Section 3.2 does the same for 2-PCSP. Section 3.3 proposes a competitive design that chooses among underlying predictors and describes an opportunity study of this approach.

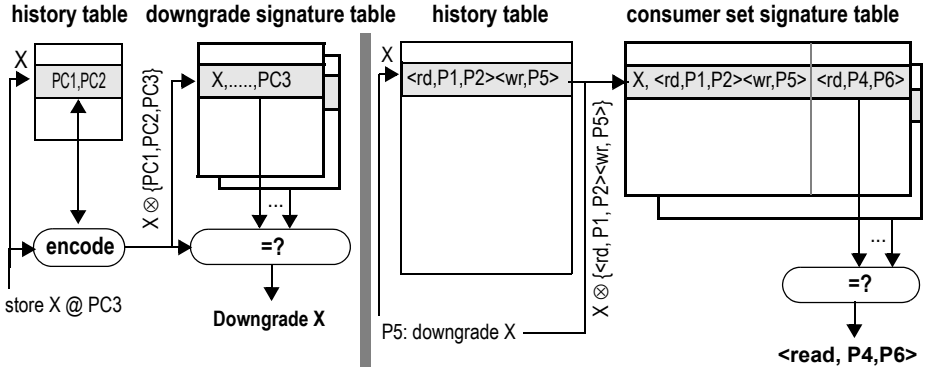

Figure 1. The 2-TDGP (left) and 2-PCSP (right) Designs

3.1 Downgrade Predictor

We propose the 2-level Trace-based DownGrade Predictor (2-TDGP), derived from the Last Touch Predictor (LTP) [11], to predict shared block downgrades. 2-TDGP predicts the last store to a cache block prior to its downgrade as a result of a read by another processor. 2-TDGP maintains a trace of all store instructions from the start of a write miss (either to an invalid or read-only cache block) until a subsequent downgrade request. Traces are used to predict the last store prior to future downgrades.

Figure 1 (left) depicts the anatomy of 2-TDGP. For each shared cache block resident in the processor's caches, the history table records a fixed-size trace, encoded using truncated addition [11] of the program counter (PC) of every store to that cache block. 2-TDGP implements the history table as a duplicate copy of the cache tag arrays to ensure 2-TDGP operations do not impact the cache's critical path.

A second table maintains signatures that are used to predict downgrades. Unlike LTP, which uses per-address signature tables, 2-TDGP organizes the signature table as a global set-associative structure. Signatures are generated by hashing bits of the block address into the history trace via an xor operation, which distributes signatures over sets in the table and improves prediction accuracy. 2-TDGP achieves much of the benefit that has been demonstrated for per-address signatures [11] with reduced storage requirements.

When an explicit downgrade message is received, 2-TDGP records the current signature for the corresponding cache block in the signature table. Each time a store instruction touches a cache block, the cache block's updated signature is looked up in the signature table. If present, 2-TDGP initiates a self-downgrade, releasing write permission and updating main memory. Each signature table entry uses a 2-bit confidence counter to add hysteresis to the training process. In the event of a misprediction, the directory detects that a self-downgrade was followed by a write by the same node, and notifies the 2-TDGP, lowering confidence of the associated signature table entry.

Figure 1 (left) depicts an example prediction. Previously, between a shared miss and downgrade, the instructions with program counters {PC1,PC2,PC3} stored to the block at address X. The history table entry indicates that {PC1,PC2} is the current

trace of stores to the block. The next store (PC3) updates the history table entry, which causes 2-TDGP to generate a signature, and look it up in the signature table. Because the table indicates a match and the 2-bit counter for the entry has saturated (not shown), 2-TDGP triggers a self-downgrade.

Much like LTP, 2-TDGP relies on repetitive program behavior to predict timely downgrades. Trace-based predictors require history table accesses to be in program order [12], because out-of-order updates to the history table disrupt the repetitive nature of signatures. 2-TDGP only records store references, thus exploiting the in-order placement of stores in the store buffer. In contrast, LTP needs auxiliary reordering mechanisms. By not recording load references, 2-TDGP minimizes the number of signatures generated and reduces predictor storage requirements compared to LTP.

A correct DGP prediction reduces the coherent read latency for the first consumer of a particular block, by the traversal time of the network. Updated data already resides in main memory; thus no downgrade is required, saving one network hop. A DGP misprediction requires the node to re-obtain write permission for the block. 2-TDGP exploits a relaxed memory system to hide the latency of these extra upgrades [1], and can therefore tolerate higher misprediction rates than LTP, for which mispredictions may result in read misses that lie on the critical path of the processor.

3.2 Pattern-Based Consumer Set Predictor

We propose the 2-level Pattern-Based Consumer Set Predictor (2-PCSP) to predict subsequent consumers of cache blocks that have been modified. 2-PCSP is derived from the Memory Sharing Predictor (MSP) [10]. Figure 1 (right) illustrates the anatomy of 2-PCSP. As in MSP, a history table maintains a history of read/write sequences for each cache block. 2-PCSP encodes history entries to be either a read or a write. Write entries contain a processor ID and read entries contain a bit vector that indicates the consumers of the previous write. The vector encoding of reads helps eliminate mispredictions due to reordering of read requests in the system [10]. We embed the history table in the directory as a register per cache block. A signature table maintains the predicted consumer set for each history pattern, using a 2-bit confidence counter per node. Each time a history pattern recurs and a particular node consumes the block, its confidence counter is increased. If the consumer does not request the block, its confidence counter is decreased.

The predictor is trained each time a write request is received by the directory. 2-PCSP encodes the current sharers of the block (i.e., immediately prior to the write being serviced by the directory) into a read entry. This read entry, and an entry for the incoming write, are appended to the current history for the block. The updated history can be looked up in the signature table and used to make a prediction. However, a prediction at this point in time is pointless, because the writing processor has not finished updating the block.

2-PCSP is best used in conjunction with a downgrade predictor. When a DGP-initiated downgrade arrives at the directory, 2-PCSP is consulted to predict subsequent sharers, using the current history. This prediction can then be used to forward the block to consumers. Although details of forwarding mechanisms are beyond the scope of this work, generalizations may be made about the effect of CSP predictions. Correctly

predicted consumers will find the block in their local memory hierarchy, thus converting a coherent read miss requiring at least one network round trip (hundreds or thousands of cycles) into a local hit (tens of cycles or less). Mispredicted consumers increase utilization of the network, both to erroneously forward the block and to invalidate it upon the next write. However, little additional latency is incurred, because invalidations of mispredicted sharers occur in parallel with invalidations of actual sharers (predicted or not).

Similar to 2-TDGP, and unlike MSP, 2-PCSP organizes the signature table as a set-associative cache, and generates signatures through an xor operation of address bits with the current history. Including address information in 2-PCSP signatures has been shown to improve prediction accuracy on scientific applications [8]. 2-PCSP's ability to capture sharing patterns also depends on the number of history entries encoded in signatures. The history depth can be arbitrarily increased to improve prediction accuracy at the cost of higher learning time and increased storage for history information.

Figure 1 (right) depicts an example prediction for a consumer set, given a history depth of two requests. For the block at address X, if a write from processor P5 is preceded by reads from P1 and P2, 2-PCSP predicts that nodes P4 and P6 will consume the produced value, as their confidence counters (not shown) have saturated.

As with 2-TDGP, our 2-PCSP design assumes that a relaxed memory system is used to tolerate write miss latency, obviating the need to predict writers. This reduces 2-PCSP prediction storage requirements by at least a factor of two over MSP. A relaxed memory system also hides the latency of invalidating incorrectly forwarded blocks, strengthening the argument given above that CSP mispredictions do not impact the critical path of the system. In contrast, incorrectly forwarding writable blocks using MSP may prematurely take readable copies away from current sharers, incurring orders of magnitude higher misprediction penalties.

3.3 Opportunity for Competitive Predictors

Our results (shown in Section 5.1 and Section 5.2 for commercial applications) demonstrate empirically that there is no optimal configuration for the number of address bits the predictors use to generate signatures. Instead, the best-performing configuration varies across target applications and operating systems. This variability in performance suggests that a more complex competitive predictor might be able to adapt to the varying application demands, and perhaps even outperform the best fixed-bit predictor configuration.

Many advanced predictors are composed of distinct base predictors with different configurations; each is better at predicting a different subset of events in the system. An algorithm or another predictor is used to choose among the available options at each prediction opportunity. The tournament branch predictor [9] used in Alpha 21264 is likely the most well known predictor employing this design. In the context of multiprocessors, R-NUMA [5] proposed a competitive algorithm for choosing between coherence mechanisms.

We present a study to determine the opportunity available to improve coverage with a competitive predictor that can use different address-bit encodings across signatures. The potential coverage of a competitive predictor is limited by the aggregate

Table 1: Applications and configurations

Scientific applications	
barnes	*64K particles., 2.0 subdiv. tol., 10.0 fleaves*
em3d	*400K nodes, 15% remote, degree 2, span 5*
moldyn	*19652 molecules, max interactions 2560000*
ocean	*514x514 grid, 9600 sec*
water	*4913 molecules, spatial, 6.2128Å cutoff*
Commercial applications	
DB2 Solaris	*TPC-C, 100 warehouses (10 GB), 96 clients, 360 MB buffer pool*
DB2 Linux	*TPC-C, 100 warehouses (10 GB), 96 clients, 450 MB buffer pool*
JBB Linux	*SPECjbb2000, 8 warehouses (200MB), 768MB Java heap*
JBB Solaris	*SPECjbb2000, 16 warehouses (400MB), 1GB Java heap*
Web Linux	*SPECweb99, Apache 2.0.48*
Web Solaris	*SPECweb99, Apache 1.3.27*

coverage achieved by all underlying predictors. We evaluate this opportunity assuming an oracle mechanism to select among the available address-bit sizes. We predict using four address-bit sizes in parallel, and after the outcome of each prediction is known, we choose the best. Results of the study are presented in Section 5.4.

4 Experimental Methodology

This study, for the first time, presents results of these prediction mechanisms on commercial applications. We compare results to previously studied scientific applications [8,10,11]. We analyze full-system memory traces created using *SimFlex* [6] on Virtutech *Simics* [15]. *Simics* is a full system simulator that allows functional simulation of unmodified commercial applications and operating systems. The simulation models all memory accesses that occur in a real system, including all OS references. We evaluate commercial workloads on Solaris 8 on SPARC and Red Hat Linux 7.3 on x86. We use two platforms because OS code has a significant impact on the performance of commercial workloads [3]. In particular, database management systems use different low-level libraries for locking and synchronization on each platform, causing distinct memory sharing behavior. We simulate a 16-node SPARC system and an 8-node x86 system (*Simics* uses a BIOS that does not support more than eight processors for x86), both with 1GB of memory. For the scientific applications, we configure Simics to simulate a 16-node multiprocessor system running Solaris 8.

Table 1 lists the applications studied and their inputs. We select a representative group of pointer-intensive and array-based scientific applications from previous studies to act as a baseline for comparison with our commercial workloads. We choose scientific applications: (1) that are scalable to large data sets, and (2) maintain a high sensitivity to memory system performance when scaled. These include *barnes* [20] a

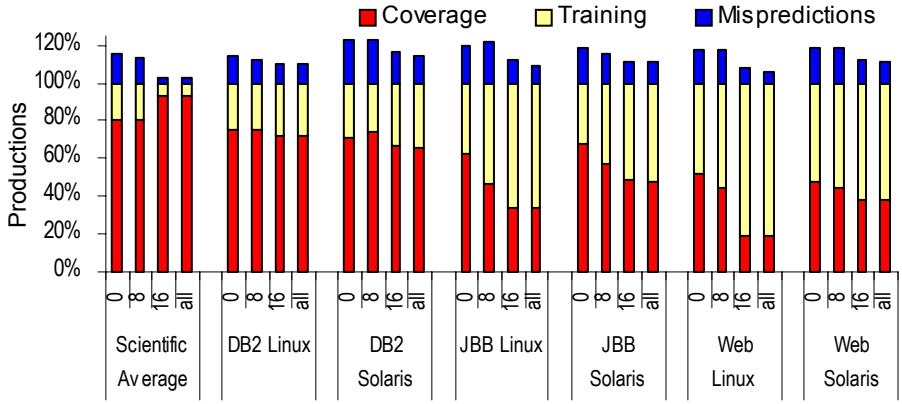

Figure 2. 2-TDGP Results for Commercial Applications. The number of data address bits used to disambiguate PC traces is indicated below each bar

hierarchical N-body simulation, *em3d* [4] an electromagnetic force simulation, *moldyn* [17] a CHARMM-like molecular dynamics simulation, *ocean* [20] current simulation, and *water* [20] an N-body molecular simulation using a 3D spatial grid.

We model the selection and design of our commercial workloads on [2]. We run version 7.2 of IBM *DB2* with the TPC-C workload [14], an online transaction processing workload. We use a highly optimized toolkit, provided by IBM, to build the TPC-C database and run the benchmark. This toolkit provides a tuned implementation of the TPC-C specification for optimal transaction execution on DB2. Prior to measurement, we warm the database until the transaction completion rate in *Simics* reaches steady state. We analyze traces of at least 5,000 transactions.

*SPECjbb*2000 [19] is a Java-based OLTP benchmark that models typical business use of a Java server application: the middle-tier of a 3-tier electronic commerce application, with a memory-resident database backend similar in design to the TPC-C database. There is a one-to-one correspondence between clients and warehouses in *SPECjbb*.

*SPECweb*99 [18] evaluates the performance of WWW servers, including both dynamic and static content. We run the *SPECweb*99 workload on recent versions of the *Apache* web server on each platform. We use a default installation of *Apache*.

5 Results

In Section 5.1 we evaluate 2-TDGP on commercial workloads, and do the same for 2-PCSP in Section 5.2. Section 5.3 compares 2-TDGP against a timer-based DGP design, and 2-PCSP against other address-based consumer predictors. We present an opportunity study in Section 5.4 that investigates the potential for competitive predictors to improve upon the base predictor designs.

5.1 Evaluation of 2-TDGP on Commercial Workloads

Figure 2 presents the coverage and mispredictions of 2-TDGP for commercial workloads, as well as an average of the scientific benchmarks studied. The perfor-

mance of 2-TDGP is measured with respect to productions (i.e., the last store a node makes to a shared cache block prior to consumption by another node). Because we normalize results to productions, these results are independent of cache size or configuration—productions and consumptions always incur coherence misses in a system without predictors. *Coverage* is the fraction of all productions that 2-TDGP predicts correctly. *Mispredictions* are placed above the 100% mark because they do not correspond to production events in a system without a DGP. Rather, they are erroneously triggered self-downgrade events. Unpredicted downgrades are labelled as *Training* because the predictor uses these to generate new signatures that can be subsequently used to predict.

We evaluate 2-TDGP with unbounded signature table storage, in order to eliminate any sensitivity to dataset size when comparing across applications. We have empirically determined that a practical finite implementation of 2-TDGP (64k entries, 16-way associativity) attains almost the same performance. We use the full PC of each store instruction in generating the PC trace (the precise number of bits varies across architectures). We vary the number of data address bits from 0 (no disambiguation) to the maximum available (26 bits for the systems studied).

Ideally, program behavior itself should be sufficient to predict memory access patterns. 2-TDGP captures this behavior through its PC history trace. However, the PC trace may not be sufficient to exactly determine program context. In particular, data address bits in 2-TDGP signatures enable the predictor to disambiguate subtrace aliases [8,11]. For example, suppose the cache blocks in a large array are each stored to five times prior to downgrade. If the array is not aligned on a cache block boundary, the first and last cache blocks may be stored only twice. The trace containing only two stores is a subtrace of the five store general case. Without additional information, the predictor cannot determine whether to predict a downgrade when this two-store subtrace is encountered. Including data address bits in the signatures distinguishes such corner cases, enabling 2-TDGP to predict correctly in each case.

Including more address bits increases 2-TDGP learning time, because particular signatures occur less frequently. Scientific applications generally have repetitious memory access patterns, so many address bits do not inhibit 2-TDGP's ability to train. Computation typically proceeds in iterations, warming 2-TDGP by the end of the second iteration (because of the 2-bit confidence counters). Subsequent iterations require no training, thus yielding very high coverage, and mispredictions are low because of subtrace disambiguation.

Commercial workloads do not exhibit the same degree of repetitiveness in their memory access patterns, because of complex datasets that continually change throughout execution. A number of programming practices common in commercial applications cause the evolution of the dataset over time, including dynamic memory allocation and garbage collection. If data address bits are included, these programming practices prevent 2-TDGP from applying signatures learned in one program context to the next. Therefore, adding address bits reduces the number of predictions made, resulting in lower coverage.

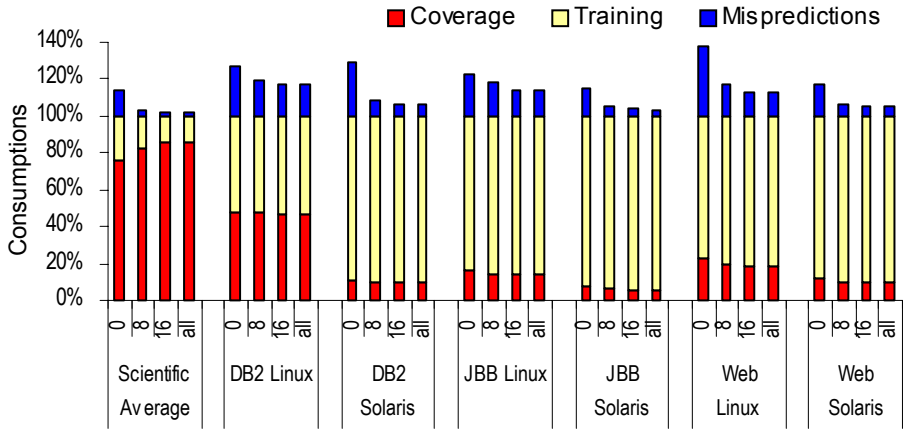

Figure 3. 2-PCSP Results for Commercial Applications. The number of data address bits used to disambiguate history signatures is indicated below each bar

5.2 Evaluation of 2-PCSP on Commercial Workloads

Figure 3 explores 2-PCSP in a similar manner. Its performance is measured with respect to consumptions (i.e., the first read by each node of a newly produced value). *Coverage* is the fraction of all subsequent readers that were predicted correctly. *Mispredictions* are nodes that were predicted to consume a value, but did not. *Training* is the gap between coverage and mispredictions, containing unpredicted patterns from which 2-PCSP learns. As with 2-TDGP, we evaluate using an infinite signature table. We have determined that a practical finite implementation of 2-PCSP (64k entries, 16-way associativity) attains similar performance. We present results for a history depth of four. In these experiments, we supply 2-PCSP with oracle knowledge of each production (i.e., as if a perfect DGP were used).

Unlike most scientific applications, commercial workloads do not exhibit regular sharing patterns. The commercial applications studied are synchronized using locks. Thus, the consumer of a particular data item depends on which CPU next acquires the lock for a critical section, which is often timing or data dependent. Data migration patterns are therefore irregular and unpredictable. CSP techniques in general, including 2-PCSP, perform well on scientific applications, as the sharing patterns, though sometimes complex, are highly repetitive [10,8].

2-PCSP achieves nearly 50% coverage for *DB2* on Linux, in stark contrast to its behavior on the other commercial workloads. This is caused by 2-PCSP's ability to accurately predict sharing of highly contended spin locks in the Linux kernel. When several nodes are spinning on a lock, 2-PCSP identifies the nodes and correctly predicts their consumption of the lock variable. The contention on such a lock generates considerable coherence traffic. Together, these two factors lead to unusually high coverage. For *DB2* on Linux, the increased 2-PCSP coverage is caused almost entirely by the kernel lock *io_request_lock*. This is due to heavy I/O activity in online transaction processing, both to retrieve database pages and to write log entries for each transaction.

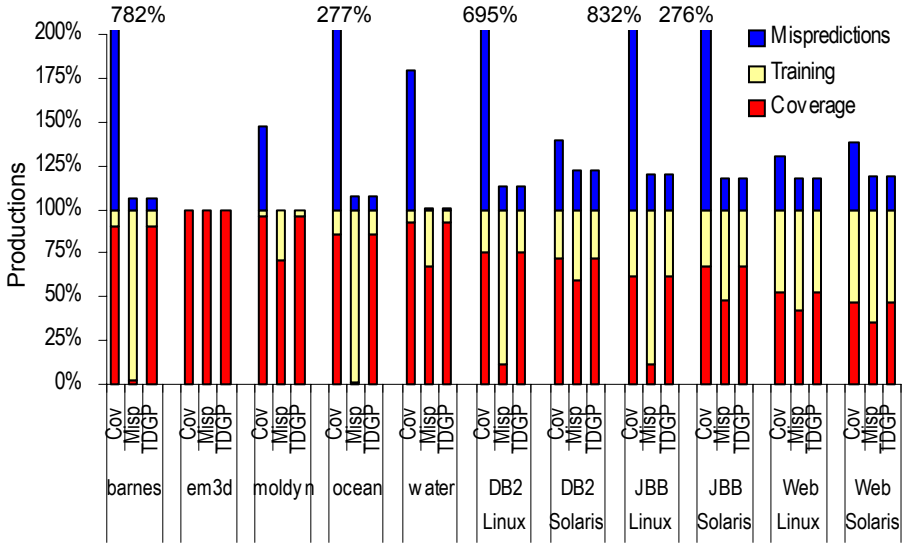

Figure 4. Comparison of DGP Techniques

For scientific applications, adding address bits to disambiguate 2-PCSP signature aliases uniformly improves both coverage and mispredictions. As with 2-TDGP, adding address or other information can eliminate trace aliases [8]. However, this is not the case for commercial workloads. All commercial workloads show the highest coverage with no address disambiguation. Removing address bits allows the predictor to reuse sharing patterns learned for one address on another, increasing the number of predictions that can be made. However, lack of disambiguation causes many of the additional predictions to be incorrect, leading to increased mispredictions. The inherent non-repetitiveness of sharing patterns in commercial workloads is the root cause of this phenomenon.

5.3 Comparisons with Other Techniques

We compare 2-TDGP against a simple timer-based DGP, similar to the dead-time predictor found in [7]. Every cache block possesses a timer, which is reset on each store to the block. Timers are decremented on every access to the cache, and on timer expiration, we predict the corresponding block should be self-downgraded. For each workload, we evaluate performance for a wide range of timer values, and present two results: one that matches 2-TDGP's coverage (labelled *Cov*) and one that matches 2-TDGP's misprediction rate (labelled *Misp*). Figure 4 presents the results. *TDGP* represents a realistic configuration, with 64k signature entries and 16-way associativity.

The nature of the timer-based DGP allows it to attain any coverage (at the expense of mispredictions) or misprediction rate (at the expense of coverage). The workloads studied exhibit a range of timer values for which coverage and mispredictions are both reasonable. Outside this range, mispredictions jump to hundreds of percent or coverage drops below 10%, depending on whether the timer value is decreased or increased. In matching with 2-TDGP, there is evidence of all three scenarios.

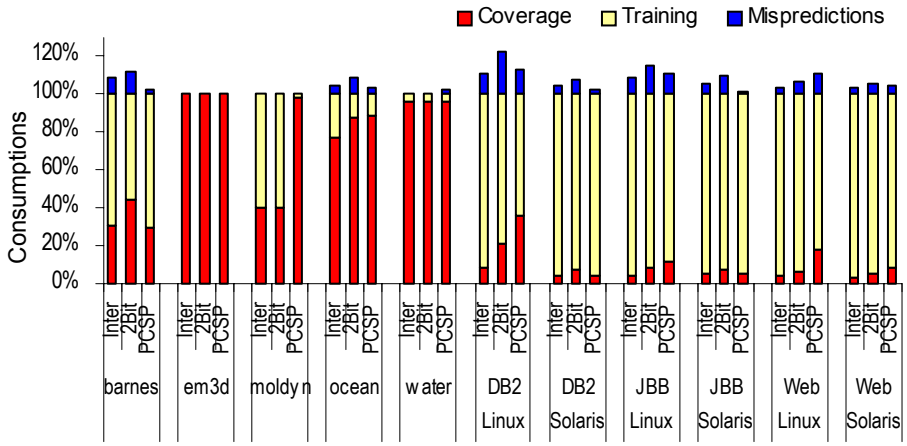

Figure 5. Comparison of CSP Techniques

There are no workloads for which the timer-based DGP performs better than 2-TDGP. A timer cannot learn sufficiently complex behavior to exhibit high coverage and low mispredictions simultaneously. Indeed, the large variability in performance of the timer-based DGP suggests its base design be re-evaluated.

Figure 5 presents a comparison of 2-PCSP with two other consumer set prediction mechanisms. *Inter* records the previous two consumer sets for a cache block, and predicts the intersection of the two [8]. *2Bit* keeps a 2-bit saturating counter for all nodes for every cache block (i.e., the same storage overhead as *Inter*). When a block transitions from shared state, the counter is incremented for all sharers of the cache block, and decremented for non-sharers. *PCSP* represents a realistic 2-PCSP configuration with 64k signature entries and 16-way associativity. Additionally, we have investigated using the single most recent consumer set for prediction, but do not include results because of its unacceptable misprediction rate (> 60%).

2Bit outperforms *Inter* because its hysteresis smooths over temporary perturbations in a sharing pattern. Nevertheless, neither of the simple sharing predictors can capture complex sharing patterns. For example, in *moldyn*, data is shared according to different patterns during distinct phases of each iteration. 2-PCSP's history-based approach is able to record this complex pattern in its signature table, correctly predicting each subsequent set of sharers from recent sharing history across phases. The *2Bit* and *Inter* results further demonstrate the unpredictability of sharing patterns in commercial applications.

5.4 Competitive Predictor Opportunity

Section 5.1 and Section 5.2 demonstrated that 2-TDGP and 2-PCSP are sensitive to their address bit configuration. In this section, we present the opportunity for a competitive design, which dynamically chooses the amount of address information to utilize, to outperform any of the fixed address bit designs. We evaluate a competitive predictor composed of a set of underlying fixed address bit predictors that are accessed and updated in parallel. At each prediction opportunity, the competitive predictor

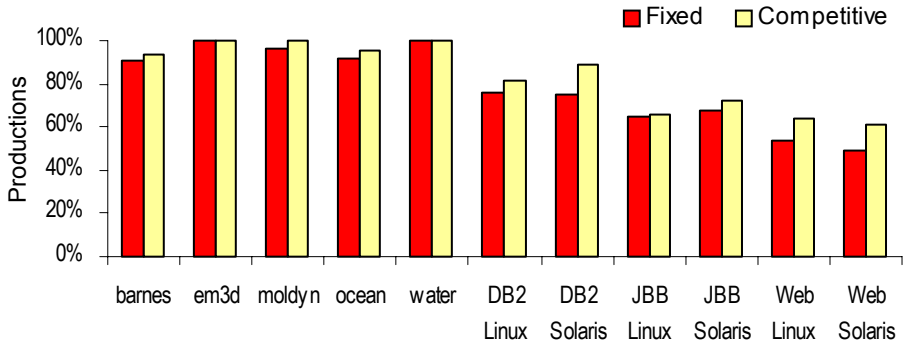

Figure 6. Competitive TDGP Opportunity. *Fixed* reproduces the best fixed-address bit result from Figure 2. *Competetive* represents an oracle competitive predictor

chooses from the available predictors. Competitive algorithms have been shown to meet the accuracy of the underlying prediction algorithms, and can even outperform the best underlying design in some cases [5]. In this paper, we present an opportunity analysis for such a hybrid, and do not consider the design of the mechanism which selects among the predictors. The results presented assume an oracle selection mechanism that chooses the predictor that will result in the highest coverage at each prediction opportunity. Because misprediction rates are highly sensitive to the selection mechanism and our oracle avoids nearly all mispredictions, we present only coverage results. We compare the best fixed design with a competitive predictor composed of the four address bit configurations presented in Section 5.1 and Section 5.2. All predictor configurations use unlimited storage.

Figure 6 presents the opportunity for the oracle competitive predictor to better 2-TDGP. The scientific applications have little room for improvement, yet the competitive predictor shows coverage gains up to 4%. Multiple predictors alleviate the effects of aliasing that are seen with a single predictor. The commercial workloads achieve up to 14% higher coverage. Those running on Solaris show a much larger increase in coverage than their Linux counterparts; this is likely due to common OS elements utilized by all applications.

Due to space constraints, we do not present a graph of the opportunity results for 2-PCSP. The improvements are minor, with no workload achieving more than 6% higher coverage. For commercial applications, even a competitive PCSP is not able to attain reasonable coverage. Given the non-repetitiveness of memory sharing patterns, this matches our expectations—regardless of the complexity of a predictor, high coverage cannot be obtained.

A competitive prediction approach has substantial opportunity to improve coverage over 2-TDGP. It is worthwhile to investigate further in this direction, to design and explore a realistic implementation. Such a design may even reduce predictor storage requirements, as it will generate fewer signatures than a base 2-TDGP design that uses many data address bits. Depending on the performance of such a predictor, it may be beneficial to also study methods for minimizing mispredictions.

6 Conclusion

In this paper, we studied two predictors for memory coherence activity in distributed shared memory architectures. 2-TDGP predicts the last store to a cache block prior to consumption by another node, and 2-PCSP predicts the consumers of updated cache blocks. We evaluated this class of predictors for the first time on commercial workloads and determined that 2-TDGP correctly predicts 47%-76% of productions, while 2-PCSP is largely ineffective due to the inherent non-repetitiveness of memory access patterns in these applications. We studied an oracle competitive predictor design, and found opportunity to increase 2-TDGP coverage up to 14%.

References

[1] S. V. Adve and K. Gharachorloo. Shared memory consistency models: A tutorial. *IEEE Computer*, 29(12):66–76, Dec. 1996.

[2] A. R. Alameldeen, C. J. Mauer, M. Xu, P. J. Harper, M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood. Evaluating non-deterministic multi-threaded commercial workloads. In *Proceedings of the Workshop on Computer Architecture Evaluation Using Commercial Workloads*, Feb. 2002.

[3] L. A. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 3–14, June 1998.

[4] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, Nov. 1993.

[5] B. Falsafi and D. A. Wood. Reactive NUMA: A design for unifying S-COMA and CC-NUMA. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 229–240, June 1997.

[6] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, May 2004.

[7] Z. Hu, S. Kaxiras, and M. Martonosi. Timekeeping in the memory system: predicting and optimizing memory behavior. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, May 2002.

[8] S. Kaxiras and C. Young. Coherence communication prediction in shared memory multiprocessors. In *Proceedings of the Sixth IEEE Symposium on High-Performance Computer Architecture*, Jan. 2000.

[9] R. E. Kessler. The alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, Mar. 1999.

[10] A.-C. Lai and B. Falsafi. Memory sharing predictor: The key to a speculative coherent DSM. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*, May 1999.

[11] A.-C. Lai and B. Falsafi. Selective, accurate, and timely self-invalidation using last-touch prediction. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[12] A.-C. Lai and B. Falsafi. Dead-block prediction & dead-block correlating prefetchers. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, July 2001.

[13] A. R. Lebeck and D. A. Wood. Dynamic self-invalidation: Reducing coherence overhead in shared-memory multiprocessors. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 48–59, June 1995.

[14] C. Levine. TPC-C: The OLTP benchmark. In *TPC Technical Report Article at www.tpc.org*.

[15] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. H. amd Johan Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.

[16] S. S. Mukherjee and M. D. Hill. Using prediction to accelerate coherence protocols. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[17] S. S. Mukherjee, S. D. Sharma, M. D. Hill, J. R. Larus, A. Rogers, and J. Saltz. Efficient support for irregular applications on distributed-memory machines. In *Fifth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming (PPOPP)*, pages 68–79, July 1995.

[18] Standard Performance Evaluation Corporation. SPECweb99. http://www.spec.org/web99/, 1999.

[19] Standard Performance Evaluation Corporation. SPECjbb2000. http://www.spec.org/jbb2000/, 2000.

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, July 1995.

[21] T.-Y. Yeh and Y. N. Patt. Two-level adaptive branch prediction. In *Proceedings of the 24th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 24)*, pages 51–61, December 1991.