

Memory Coherence Activity Prediction in Commercial Workloads

Stephen Somogyi

December 2004

Department of Electrical and Computer Engineering

Carnegie Mellon University

Pittsburgh, Pennsylvania

Submitted in partial fulfillment of the requirements

for the degree of Master of Science.

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. Together, these observations suggest the importance of investigating coherence activity prediction in the context of commercial workloads.

This thesis studies a trace-based Downgrade Predictor (DGP) for predicting last stores to shared cache blocks prior to consumption by other processors, and a pattern-based Consumer Set Predictor (CSP) for predicting subsequent readers. We evaluate this class of predictors on commercial applications and demonstrate that our DGP correctly predicts 51%-94% of last stores. Memory sharing patterns in commercial workloads are inherently non-repetitive; hence CSP cannot predict a significant fraction of consumers. We develop an improved DGP design by extending our trace-based DGP to exploit multiple underlying prediction tables, and demonstrate this hierarchical predictor accurately predicts 63%-97% of last stores on commercial workloads.

Acknowledgements

I would like to thank my advisor, Professor Babak Falsafi, for his guidance on research and keen insights into computer architecture. I would like to thank my parents and my brother, who have supported me in many ways despite being on the other side of the continent. To all my amazing friends, both at home and in Pittsburgh, I thank you too. Finally, I would like to thank members of the A-level community, in particular the Impetus group, for their companionship, loud voices, encouragement, and many extended discussions that make research happen.

Table of Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 7 |
| 2 | Related Work | 9 |
| 3 | Design | 11 |
| 3.1 | Trace-Based Downgrade Predictor | 11 |
| 3.2 | Pattern-Based Consumer Set Predictor | 13 |
| 3.3 | Hierarchical Downgrade Predictor | 15 |
| 4 | Experimental Methodology | 17 |
| 5 | Results | 19 |
| 5.1 | Evaluation of the Trace-based Downgrade Predictor | 19 |
| 5.2 | Evaluation of the Pattern-based Consumer Set Predictor | 21 |
| 5.3 | Comparison of Downgrade Prediction Techniques | 23 |
| 5.4 | Comparison of Consumer Set Prediction Techniques | 24 |
| 5.5 | Analysis of TDGP Signature Behavior | 25 |
| 5.6 | Evaluation of the Hierarchical Downgrade Predictor | 28 |
| 6 | Conclusion | 31 |

List of Figures

| | | |
|-----------|--|----|
| Figure 1. | The Trace-based Downgrade Predictor Design..... | 11 |
| Figure 2. | The Pattern-based Consumer Set Predictor Design..... | 13 |
| Figure 3. | The Hierarchical Trace-based Downgrade Predictor Design..... | 16 |
| Figure 4. | TDGP Performance | 20 |
| Figure 5. | PCSP Performance | 22 |
| Figure 6. | Comparison of DGP Techniques..... | 23 |
| Figure 7. | Comparison of CSP Techniques..... | 25 |
| Figure 8. | Hierarchical TDGP Performance | 28 |

List of Tables

| | | |
|----------|--|----|
| Table 1. | Evaluated Applications and Configurations | 17 |
| Table 2. | TDGP Signature Details for Commercial Workloads | 26 |
| Table 3. | Signature Reduction for Scientific Applications under H-TDGP | 29 |

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 with additional communication channels, which are typically straightforward to implement, albeit costly. Unfortunately, interconnect latency cannot be easily reduced, regardless of cost. Although microarchitectural and memory coherence optimizations such as out-of-order execution and relaxed memory models can hide some of the latency associated with coherence traffic [1], they cannot completely overlap the shared read miss latency.

This thesis studies two prediction mechanisms that can be used to reduce coherence latency in DSM. We derive these predictors from prior work [8,9], which evaluated them using scientific workloads. The Trace-based DownGrade Predictor (TDGP) identifies the final store to a cache block prior to a subsequent read by another node, and self-downgrades the block, eliminating one network hop from the ensuing coherent read request. The Pattern-based Consumer Set Predictor (PCSP) predicts which nodes will subsequently read (consume) a value that has been written (produced) by some node. Although outside the scope

of this thesis, such a prediction can be used to forward blocks to consumers, thus completely obviating the need for a coherent read request and reducing effective latency by several orders of magnitude.

Evaluations of architectural proposals continually increase in sophistication. Recently, commercial applications have become important to the research community [2]. Coherence latency has been identified as a first-order determinant of database performance in multiprocessor systems [3], and coherence traffic increases with aggregate caching in the memory hierarchy. 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 memory access traces from full-system simulation of shared-memory multiprocessors [5] running scientific and commercial workloads on stock operating systems, we demonstrate:

- **TDGP on Commercial Workloads:** We evaluate a 2-level trace-based downgrade predictor on commercial workloads and show that, despite long and complex execution paths, TDGP correctly identifies 51%-94% of productions with 7%-26% mispredictions.
- **PCSP on Commercial Workloads:** We evaluate a 2-level pattern-based consumer set predictor on commercial workloads, and show that it cannot predict a significant fraction of consumers. Because of data- and timing-dependent behavior, the sharing patterns in commercial applications are inherently non-repetitive and therefore unpredictable.
- **Hierarchical TDGP Design:** We observe that 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 develop a hierarchical TDGP design using multiple underlying predictor tables, and demonstrate improved coverage of 63%-97%.

The rest of this thesis is organized as follows. In Section 2 we review related work in the field. Section 3 presents the design of TDGP and PCSP. Section 4 describes our experimental infrastructure and procedures. Section 5 presents the results. We conclude the thesis 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 [11]. Their technique, *Dynamic Self-Invalidation*, triggers invalidation of shared data blocks, identified via coherence protocol hints, at annotated critical section boundaries. *Last Touch Prediction* (LTP), proposed in [9], instead associates invalidation events with the instruction sequence that accesses a cache block prior to its invalidation. By storing PC traces that repetitively lead to invalidation, LTP can trigger self-invalidation immediately upon the last access without program annotations. Our 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, because these techniques cannot fully overlap coherent read latency, the retrieval (via a downgrade operation) of a value modified by another node remains on the processor’s critical path. [9] evaluated LTP using scientific workloads. This thesis presents an evaluation of this class of predictor for commercial applications. In the context of a uniprocessor system, Hu et al. [6] investigated timekeeping approaches for predicting memory system events. These techniques can be adapted for coherence prediction in DSM, and in this thesis we compare TDGP with a *dead-time*-based predictor.

Predicting the subsequent consumers of a newly produced value, which we generically call consumer set prediction (CSP), was first attempted by Mukherjee and Hill in [13]. Their scheme adapts two-level branch prediction [18] to predict coherence messages based on the history of previously received messages. *Memory Sharing Prediction* (MSP) [8] 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 PCSP further optimizes MSP to predict only read requests. As with 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 [7], and classifies CSPs based on their access and prediction functions. In this taxonomy, PCSP is best categorized as address-based access with two-level prediction. However, PCSP differs in that it uses per-node saturating confidence counters in the second-level table to reduce mispredictions. Previous work on consumer set prediction has evaluated predictors using scientific applications; this thesis presents the effectiveness of PCSP for commercial workloads.

A preliminary version of the research documented in this thesis appeared in [15], including an opportunity study for competitive TDGP. Compared to [15], this thesis investigates issues in greater detail, evaluates a larger and improved set of workloads, and presents a physical design for the hierarchical trace-based downgrade predictor.

3 Design

In this section we present the details of our predictor designs. Section 3.1 presents TDGP, its derivation from LTP, and an example of its operation. Similarly, Section 3.2 derives PCSP from MSP. Section 3.3 proposes a hierarchical design for a trace-based downgrade predictor that exploits multiple underlying prediction tables.

3.1 Trace-Based Downgrade Predictor

We propose the 2-level Trace-based DownGrade Predictor (TDGP), derived from the Last Touch Predictor (LTP) [9], to predict shared block downgrades. TDGP predicts the last store to a cache block prior to its downgrade as a result of a read by another processor. TDGP maintains a trace of all store instructions following a write miss (either to an invalid or read-only cache block) until a subsequent downgrade request. Recorded traces are used to predict the last store prior to future downgrades.

Figure 1 depicts the anatomy of TDGP. For each shared cache block resident in the processor’s caches, the history table records a fixed-size trace, encoded using truncated addition [9], of the program counter (PC) of every store to that cache block. TDGP implements the history table as a duplicate copy of the cache tag arrays to ensure TDGP operations do not impact the cache’s critical path.

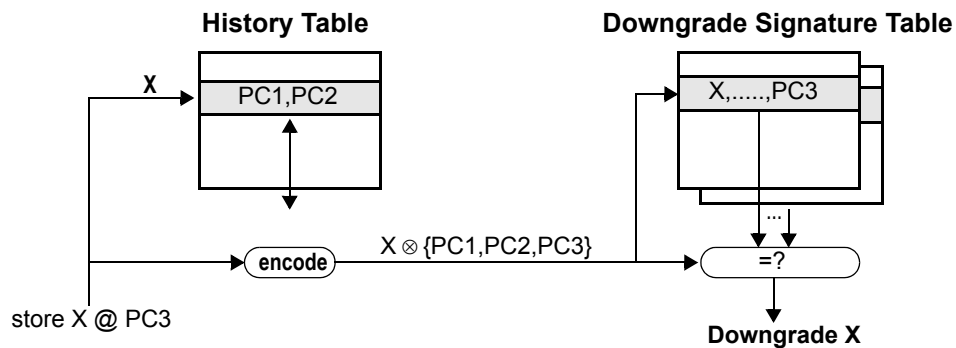


Figure 1. The Trace-based Downgrade Predictor Design

A second table maintains signatures that are used to predict downgrades. Unlike LTP, which uses per-address signature tables, 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. TDGP achieves much of the benefit that has been demonstrated for per-address signatures [9] with reduced storage requirements.

When an explicit downgrade message is received, 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 the signature is found, 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 coherence directory detects that a self-downgrade was followed by a write by the same node, and notifies the TDGP, lowering confidence of the associated signature table entry.

Figure 1 also depicts an example prediction. Previously, between a shared miss and downgrade, the instructions with program counters PC1, PC2 and PC3 performed stores 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 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 high confidence (not shown), TDGP triggers a self-downgrade.

Much like LTP [9], TDGP relies on repetitive program behavior to predict timely downgrades. Trace-based predictors require history table accesses to be in program order [10], because out-of-order updates to the history table disrupt the repetitive nature of signatures. 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, TDGP minimizes the number of signatures generated and reduces predictor storage requirements compared to LTP.

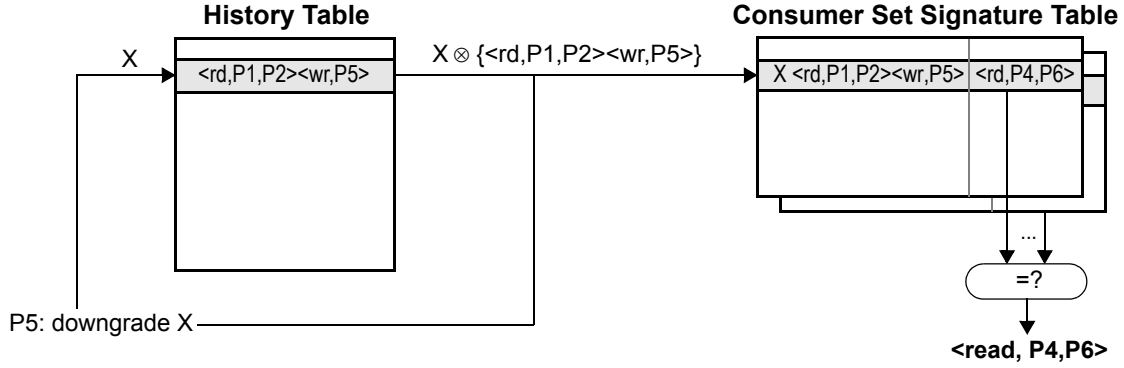


Figure 2. The Pattern-based Consumer Set Predictor Design

A correct DGP prediction reduces the coherent read latency for the first consumer of a particular block, by the traversal time of the network. Because updated data already resides in main memory, the directory need not explicitly retrieve the data from the updating node, thus saving one network hop. A DGP misprediction requires the node to re-obtain write permission for the block. 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 are on the critical path of the processor.

3.2 Pattern-Based Consumer Set Predictor

We propose the 2-level Pattern-Based Consumer Set Predictor (PCSP) to predict subsequent consumers of cache blocks that have been modified. PCSP is derived from the Memory Sharing Predictor (MSP) [8]. Figure 2 illustrates the anatomy of PCSP. As in MSP, a history table maintains the read/write sequence for each cache block. 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 [8]. The history table is embedded 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, the confidence counters for consuming nodes are increased. For nodes that do not request the block, their confidence counters are decreased.

The predictor is trained each time a write request is received by the directory. 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 instant in time is premature, because the writing processor has not finished updating the block.

PCSP can be used in conjunction with a downgrade predictor. When a DGP-initiated downgrade arrives at the directory, 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 forwarded 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).

Similarly to TDGP, 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 PCSP signatures has been shown to improve prediction accuracy on scientific applications [7]. 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 longer training time and increased storage for history information.

Figure 2 also 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, PCSP

predicts that nodes P4 and P6 will consume the produced value, as their saturating counters (not shown) have high confidence.

As with TDGP, our PCSP design assumes that a relaxed memory system is used to tolerate write miss latency, obviating the need to predict writer nodes and reducing PCSP storage requirements by at least a factor of two over MSP. A relaxed memory system also hides the latency of invalidating incorrectly forwarded blocks, further indicating 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 Hierarchical Downgrade Predictor

Our results (shown in Section 5.1) demonstrate empirically that there is no optimal configuration for the number of address bits TDGP uses to generate signatures. Instead, the best-performing configuration varies across target workloads, and even within a particular application, no single configuration captures all of the TDGP opportunity. This variability suggests that a more complex predictor might be able to adapt to the varying application demands, and therefore outperform the baseline TDGP design.

We propose the Hierarchical Trace-based DownGrade Predictor (H-TDGP) to exploit multiple underlying prediction tables that are each best suited for capturing different aspects of program behavior. As shown in Figure 3, H-TDGP maintains a single history table, but two signature tables: a zero-bit table for signatures that require no address disambiguation, and a high-bit table (e.g., using 8 or 16 address bits) for signatures that benefit from the inclusion of address information.

The history table is identical, both in structure and access, to that in the base TDGP design. On a store by the processor, the current PC is encoded into the existing history trace, and this updated trace is written back to the history table. The trace is then used directly to look up in the zero-bit table, whereas an xor operation hashes the trace with bits from the address and the resultant signature is looked up in the high-bit table.

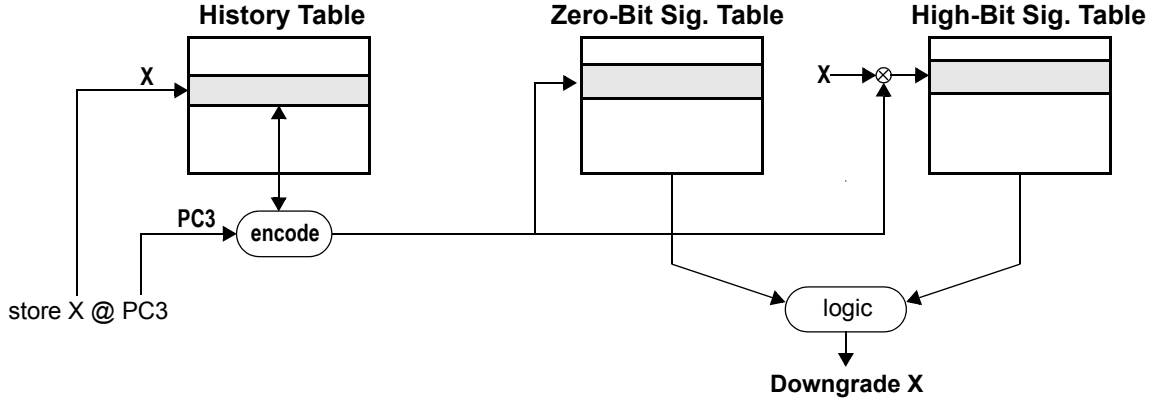


Figure 3. The Hierarchical Trace-based Downgrade Predictor Design

Although the structure of each signature table is identical to that in the base TDGP design, the policies through which they are accessed and updated are different. We prefer to store signatures that do not require address disambiguation in the zero-bit table, to reduce storage requirements. Therefore, on the first occurrence of a particular PC-trace, the corresponding signature is placed in the zero-bit table. If a signature does not lead to mispredictions, there is no need to place it in the high-bit table. However, when H-TDGP detects that a signature in the zero-bit table causes mispredicted downgrades, the corresponding PC-trace is promoted to the high-bit table. On subsequent downgrades with the same history trace, the corresponding signature will be placed in the high-bit table.

As in the base TDGP, saturating counters maintain confidence for all signatures (in both tables). We use the minimum value to indicate a PC-trace needs promotion, as described above. However, we implement slightly different update and prediction policies for the scientific versus the commercial applications. Scientific code is very structured and repetitive; if a signature ever mispredicts, it will likely always benefit from address disambiguation. In commercial workloads, different program behaviors are more common, and the pattern of these behaviors might not be repetitive. Therefore, with commercial applications, H-TDGP always updates the confidence counters of the signatures in both tables (if the signature is present), and will predict a downgrade if either signature has high confidence. With scientific applications, H-TDGP gives priority to signatures in the high-bit table, and only looks in the zero-bit table if the signature is not present in the high-bit table.

4 Experimental Methodology

This thesis presents results of these coherence prediction mechanisms on commercial applications. For completeness, we compare results to previously studied scientific applications [7,8,9]. We analyze full-system memory traces, created using SIMFLEX [5] on Virtutech *Simics* [12], of a 16-node system with 3 GB of RAM running Solaris 8. *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 OS references. This is necessary because OS code has a significant impact on the performance of commercial workloads [3].

Table 1 describes the applications and parameters used for evaluation in this thesis. We target our study at commercial workloads, but include a representative group of pointer-intensive and array-based scientific applications for comparison. We choose scientific applications that are (1) scalable to large data sets, and (2) maintain a high sensitivity to memory system performance when scaled. We include *barnes* [17], a hierarchical N-body simulation, *em3d* [4], an electromagnetic force simulation, *molodyn* [14], a CHARMM-like molecular dynamics simulation, and *ocean* [17] current simulation.

Table 1: Evaluated Applications and Configurations

| Scientific applications | |
|--------------------------------|--|
| <i>barnes</i> | 64k particles, 2.0 subdivision tolerance, 10.0 fleaves |
| <i>em3d</i> | 400k nodes, 15% remote, degree 2, span 5 |
| <i>molodyn</i> | 19652 molecules, 2.5M max interactions, boxsize 17 |
| <i>ocean</i> | 514x514 grid, 9600s relaxation, 20k res., error tolerance 1e-7 |
| Commercial applications | |
| <i>Apache</i> | SPECweb, 2000 connections, default server configuration |
| <i>DB2</i> | TPC-C, 100 warehouses (10 GB), 400 clients, 450 MB buffer pool |
| <i>JBB</i> | SPECjbb, 16 warehouses (400 MB), 1 GB Java heap |
| <i>Zeus</i> | SPECweb, 2000 connections, default server configuration |

We evaluate the *IBM DB2 v7.2 EEE* database management system running the TPC-C v3.0 online transaction processing workload. We use an optimized toolkit, provided by IBM, to build and populate the TPC-C database. The toolkit provides a tuned implementation of the TPC-C specification. User think times are set to zero and each user starts a separate database connection. Database data are partitioned across multiple disks and the log is located on dedicated storage.

We evaluate the performance of WWW servers running the SPECweb99 benchmark on a default installation of *Apache HTTP Server v2.0* and *Zeus Web Server v4.3*. We evaluate each web server with two different SPECweb configurations, one biased towards dynamic web content (40% dynamic GETs), and the other towards static content (90% static GETs). In the results, we append “Dynamic” or “Static” to the web server names to indicate the dynamic and static configurations, respectively.

SPECjbb2000 [16] 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.

The trace-based analyses in this thesis use memory access traces collected from SIMFLEX with in-order execution, no memory system stalls and a fixed IPC of 1.0. We analyze traces of at least ten iterations for scientific applications. We warm commercial applications for at least 5,000 transactions/web requests prior to starting traces, and then trace at least 500 transactions/requests. We use the first iteration of each scientific and first 100 million instructions (per CPU) of each commercial application to warm trace-based simulations prior to measurement.

5 Results

In Section 5.1 we evaluate TDGP on commercial workloads, and do the same for PCSP in Section 5.2. Section 5.3 compares TDGP against a timer-based DGP design, while Section 5.4 compares PCSP to other address-based consumer predictors. We investigate TDGP signature behavior for commercial workloads in Section 5.5 and present an evaluation of Hierarchical TDGP in Section 5.6.

5.1 Evaluation of the Trace-based Downgrade Predictor

Figure 4 presents the coverage and mispredictions of TDGP for the benchmarks studied. The performance of 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 TDGP predicts correctly. *Mispredictions* are indicated 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 update confidence or generate new signatures that can be subsequently used to predict.

We evaluate TDGP with unbounded signature table storage, in order to eliminate sensitivity to dataset size when comparing across applications. We have empirically determined that a practical finite implementation of TDGP (128k entries, 16-way associativity) attains almost the same performance. We use the full PC of each store instruction in generating the PC trace. We vary the number of data address bits from 0 (no disambiguation) to 26 (the maximum available for the system studied).

Given sufficient address information, TDGP performs very well on the scientific applications, achieving very high coverage with few mispredictions. These results corroborate prior evaluation of LTP [9]. There is wide variation in the performance of TDGP on commercial workloads. While the static web

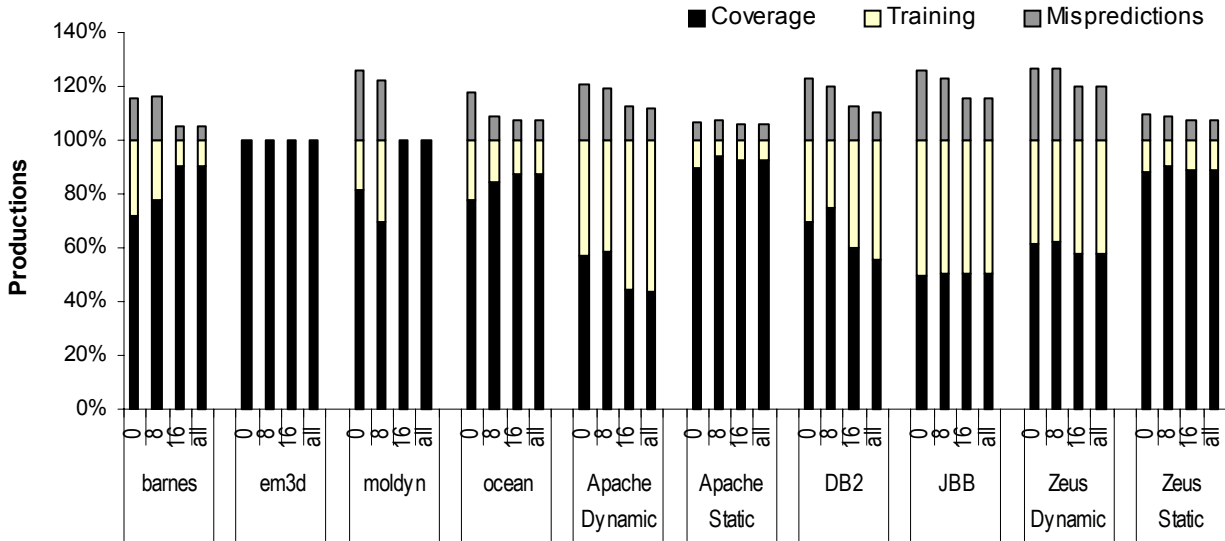


Figure 4. TDGP Performance. The number of data address bits used to disambiguate PC traces is indicated below each bar

servers achieve greater than 90% coverage and less than 9% mispredictions, coverage on the other applications ranges between 51% and 75%. Clearly, the high proportion of static requests in the static web servers leads to extremely repetitive behavior. The other commercial workloads studied have a significant portion of dynamic content/activity, and therefore behave less repetitively. As data structures evolve, so will the sequences of PCs that lead to last stores. However, a majority of productions are predictable, indicating that data structures are not constantly changing and that access sequences repeat.

Ideally, program behavior itself should be sufficient to predict memory access patterns. TDGP captures program behavior through its PC-trace. However, the PC-trace may not be sufficient to indicate program context exactly. In particular, data address bits in TDGP signatures enable the predictor to disambiguate subtrace aliases [7,9]. 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 TDGP to predict correctly in each case.

Including more address bits increases TDGP learning time because particular signatures occur less frequently. Scientific applications generally have repetitious memory access patterns, so many address bits do not inhibit TDGP’s ability to train. Computation typically proceeds in iterations, warming 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 low mispredictions, because of subtrace disambiguation.

Commercial workloads do not exhibit the same degree of repetitiveness in their memory access patterns, because of complex datasets that 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 many data address bits are included, these programming practices prevent TDGP from applying signatures learned in one program context to the next. Nevertheless, a small number of address bits (e.g., eight) improves performance relative to the zero-bit configuration. With eight address bits (and disregarding the six bits of block offset), TDGP disambiguates addresses within memory pages. Because similar data structures are likely to be placed in similar locations on different pages, signatures created for a particular cache block on one page can be applied to data at the same page offset on other pages.

5.2 Evaluation of the Pattern-based Consumer Set Predictor

Figure 5 explores PCSP’s performance on the selected workloads. 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 PCSP learns. As with TDGP, we evaluate using an infinite signature table, but have determined that a practical finite implementation of PCSP (64k entries, 16-way associativity) attains similar performance. We present results for a history depth of four entries. In these experiments, we supply PCSP with oracle knowledge of each production (i.e., as if a perfect DGP were used).

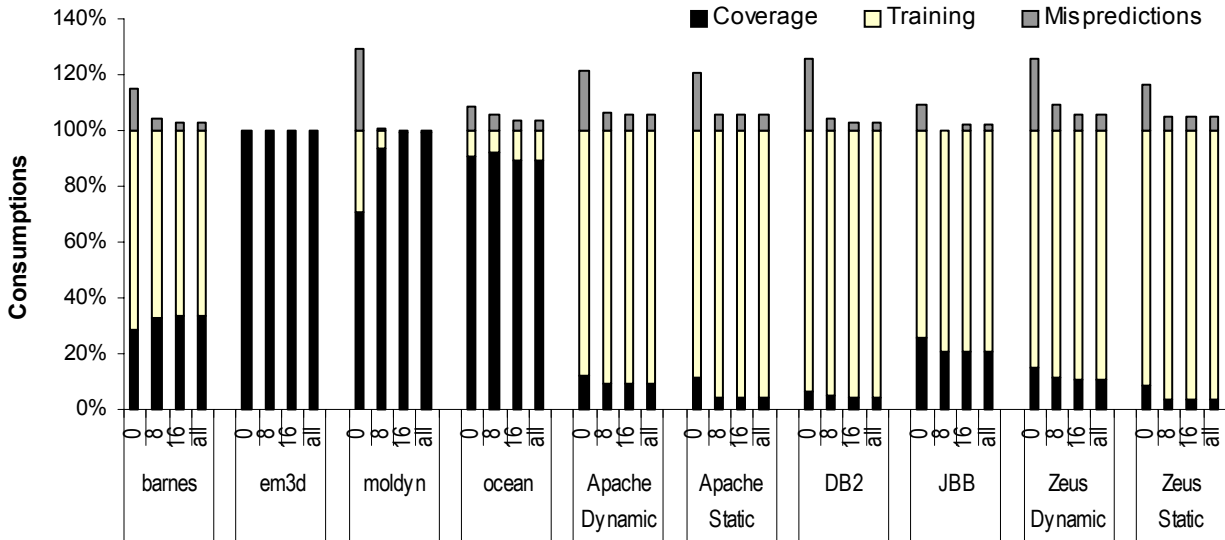


Figure 5. PCSP Performance. The number of data address bits used to disambiguate sharing histories is indicated below each bar

The commercial workloads do not exhibit any repetitiveness in their 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. For scientific applications, the PCSP results support prior work [8], with most applications exhibiting greater than 90% coverage and less than 4% mispredictions. *Barnes* is the outlier, because it uses lock-based synchronization, leading to similar behavior as the commercial applications. CSP techniques in general, including PCSP, perform well on scientific applications, as the sharing patterns, though sometimes complex, are highly repetitive [8,7].

For scientific applications, adding address bits to disambiguate PCSP signatures improves both coverage and mispredictions. As with TDGP, adding address or other information can eliminate trace aliases [7]. However, this is not the case for commercial workloads, which show the highest coverage with no address disambiguation. Removing address bits allows the predictor to reuse sharing patterns learned for one address on others, increasing the number of predictions that can be made. However, lack of disambiguation causes many of the additional predictions to be incorrect, leading to greatly increased

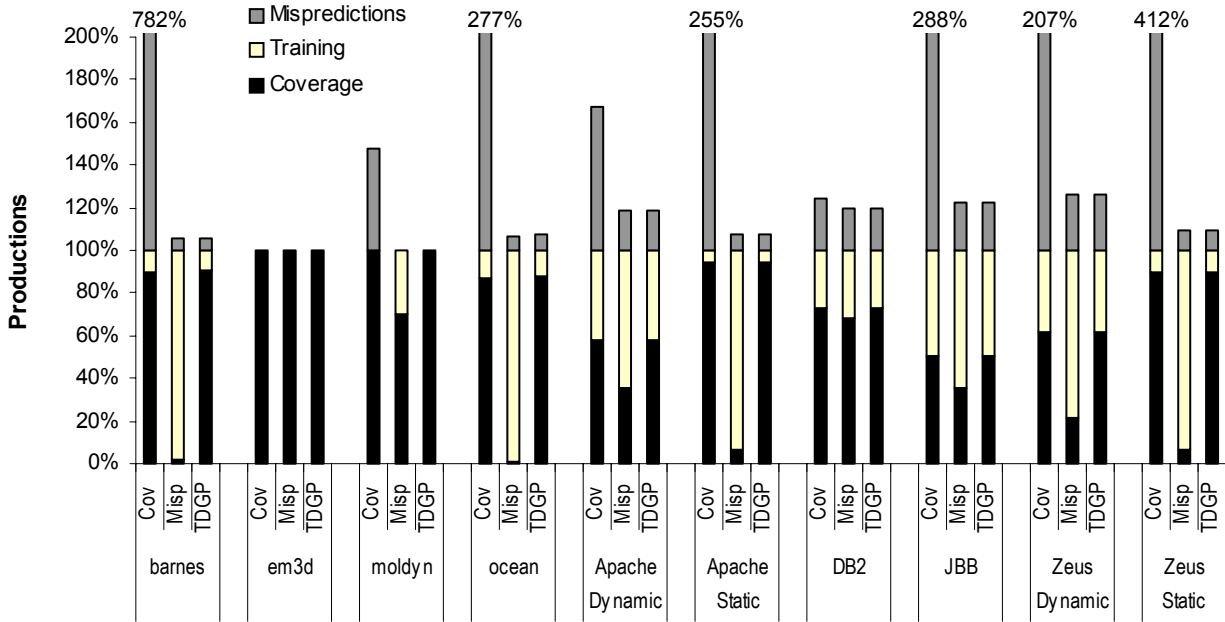


Figure 6. Comparison of DGP Techniques

mispredictions with zero address bits. The inherent non-repetitiveness of sharing patterns in commercial workloads is the root cause of this phenomenon.

5.3 Comparison of Downgrade Prediction Techniques

We compare TDGP against a simple timer-based DGP, similar to the dead-time predictor proposed in [6]. 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 in Figure 6: one configuration that matches TDGP’s coverage (labelled *Cov*) and one that matches TDGP’s misprediction rate (labelled *Misp*). *TDGP* represents a realistic configuration, with 128k 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 grow to

hundreds of percent or coverage drops to a negligible fraction of productions, depending on whether the timer value is decreased or increased. In matching with TDGP, there is evidence of all three scenarios.

The timer-based DGP performs similarly on both scientific and commercial applications, because of its ignorance of all aspects of program behavior. Although there are no applications for which the timer-based predictor performs better than TDGP, it matches TDGP’s performance on *em3d* and comes within 5% on *DB2*. In *em3d*, each shared cache block is produced exactly once per iteration, yielding a range of timer values for which the timer-based DGP performs well. In *DB2*, the timer happens to capture a majority of downgrades, and therefore performs well.

In general, a timer cannot learn sufficiently complex behavior to exhibit high coverage and low mispredictions simultaneously. To improve the timer-based DGP so it performs well on all workloads, a secondary structure/predictor may be required, which will negate the simplicity of the design. Moreover, a timer-based predictor is not capable of predicting the exact time of a last store to a cache block, which is one of the key advantages of the trace-based downgrade predictor.

5.4 Comparison of Consumer Set Prediction Techniques

Figure 7 presents a comparison of PCSP with two address-based consumer set prediction mechanisms. *Inter* records the previous two consumer sets for a cache block, and predicts the intersection of the two [7]. For every cache block, *2-Bit* keeps a 2-bit saturating counter for each node (this requires 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. Additionally, we have investigated using either the single most recent consumer set for prediction, or the union of the two most recent consumer sets, but do not include results because of their high misprediction rate (> 60%). *PCSP* represents a realistic PCSP configuration with 64k signature entries and 16-way associativity.

In the scientific applications, *2-Bit* outperforms *Inter* because its hysteresis smooths over temporary perturbations in a sharing pattern. Nevertheless, neither of the address-based predictors can capture

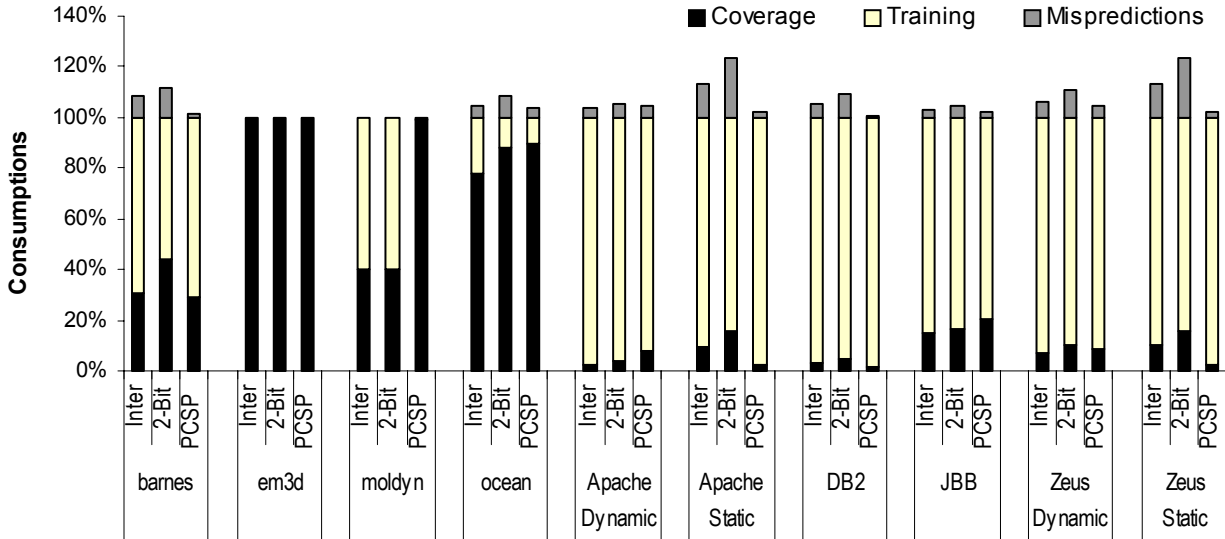


Figure 7. Comparison of CSP Techniques

complex sharing patterns, even repetitive ones. For example, in *moldyn*, data is shared according to different patterns during distinct phases of each iteration. 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. Without long-term history, other consumer set predictors are incapable of predicting these events.

The address-based predictors perform just as poorly on the commercial workloads as PCSP, achieving less than 20% coverage. *2-Bit* is better than PCSP on several applications, indicating a small amount of short term stability in sharing patterns, but not consistent enough for PCSP to capture. Fundamentally, however, these results further demonstrate the unpredictability of sharing patterns in commercial applications.

5.5 Analysis of TDGP Signature Behavior

While Section 5.2 demonstrated that consumer set prediction is not possible for commercial workloads, Section 5.1 showed that the trace-based downgrade predictor performs well on these applications, although very differently from the well-known scientific applications. This section explores the behavior of signatures created by TDGP in the context of commercial workloads.

A trace-based predictor that uses address information relies on the repetitiveness of both addresses and program control flow. In commercial applications, many addresses are not shared frequently. In other words, over an interval of execution, a particular cache block might only be produced and consumed once. A database, for example, can potentially take minutes of real time between accesses to the same page in its buffer pool. When TDGP is configured to use a large number of address bits, any reasonably sized signature table (even up to several megabytes) will have long since evicted a particular signature before it can be used to predict. Even though some data are accessed more frequently (e.g., roots of trees) and these repetitions will be learned and predicted by TDGP, significant opportunity is lost to infrequently recurring signatures.

The second column in Table 2 lists the percentage of the total number of productions for which the corresponding address is only downgraded once in the measurement interval (as described in Section 4). These account for the majority of productions in *JBB*, only a small fraction in *DB2*, and a significant fraction in the web servers. *JBB* essentially fills the entire Java heap with data, consuming most addresses only once, before performing garbage collection and repeating this cycle. *DB2*, on the other hand, is highly optimized and minimizes communication where possible, resulting in the most reuse of data. Nevertheless, for these workloads in general, any attempt to create signatures based on complete address information will squander a significant fraction of opportunity.

Table 2: TDGP Signature Details for Commercial Workloads

| Application | Single Downgrade Productions | First PCs | Single Signature PCs |
|-----------------------|-------------------------------------|------------------|-----------------------------|
| <i>Apache Dynamic</i> | 49% | 2817 | 29% |
| <i>Apache Static</i> | 39% | 2730 | 23% |
| <i>DB2</i> | 6% | 3020 | 17% |
| <i>JBB</i> | 80% | 721 | 29% |
| <i>Zeus Dynamic</i> | 39% | 3316 | 25% |
| <i>Zeus Static</i> | 26% | 2859 | 25% |

With respect to program control flow, TDGP relies on repetitive sequences of PCs that store to a cache block, between the first store after acquiring exclusive access and the last store prior to consumption by another node. In many programs, only a small subset of instructions access shared data, which TDGP inherently takes advantage of (otherwise far too many signatures would be created to fit in a reasonably-sized table). Each PC-trace begins with a single PC, and includes zero or more subsequent PCs. Because the history table attempts to uniquely encode every trace of different PCs, every different *first PC* should lead to a different set of possible histories.

The third column of Table 2 lists the raw number of first PCs encountered in each of the commercial workloads. As expected, these applications are complex and therefore a large number of instructions lead to eventual downgrades. In contrast, the scientific applications contain fewer than 100 first PCs, and as few as four in *em3d*. Surprisingly, *DB2* has a similar number of first PCs as the web servers, even though the database's code is much larger. *JBB* is the outlier because its code is more structured compared with the other commercial workloads.

Given a particular first PC, TDGP will perform best if the sequence of subsequent PCs is perfectly repetitive. However, because of the reasons described above, we expect commercial workloads to exhibit significant variability in their data access patterns. The fourth column of Table 2 lists the percentage of first PCs for which only one signature (i.e., history trace) is ever encountered. At least 70% of all PCs generate multiple signatures, which confirms the expectation above. *DB2* shows the most variability, suggesting that its control flow is more data dependent than the other applications.

Despite the challenges documented in this section, TDGP performs very well, correctly predicting at least half of all downgrades. Even though control flow is not perfectly repetitive, TDGP learns many different PC sequences and takes advantage of sequences that occur more frequently. Likewise, although addresses are not highly repetitive, TDGP achieves some degree of address disambiguation through the use of a small number of address bits.

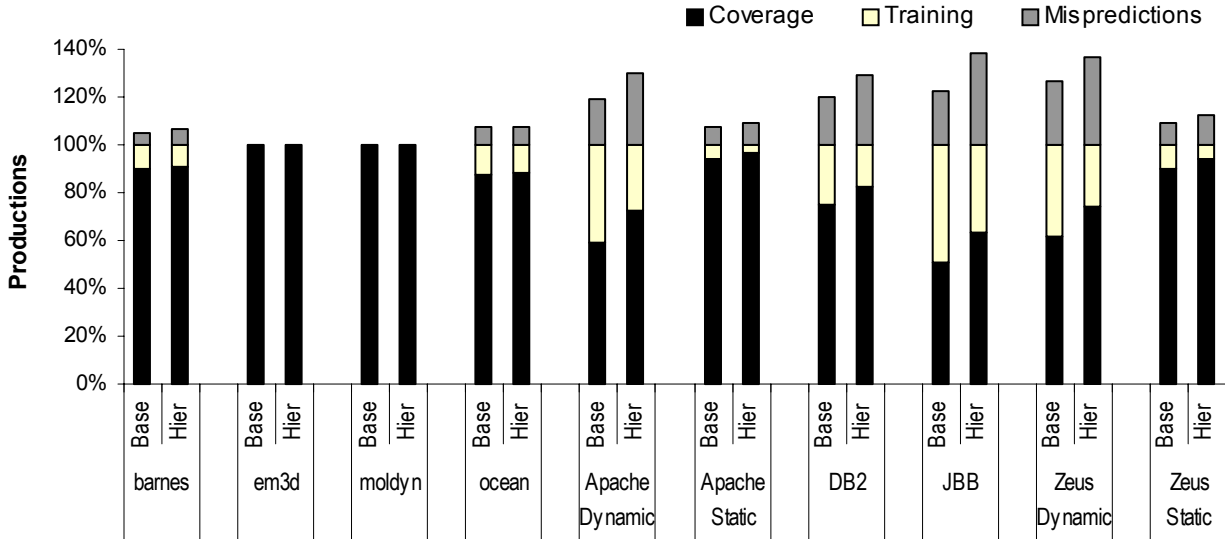


Figure 8. Hierarchical TDGP Performance

5.6 Evaluation of the Hierarchical Downgrade Predictor

Section 5.1 established that TDGP is sensitive to its address bit configuration. This observation led us to propose the hierarchical TDGP design described in Section 3.3. By exploiting two signature tables, one without address disambiguation (i.e., a zero address bit configuration) and one with, we aim to improve the performance of the base TDGP. For the scientific applications, there is little opportunity to increase coverage, but we can reduce the storage requirement by maintaining some signatures in the zero-bit table. For the commercial workloads, we can attain higher coverage by using the information contained in both signature tables to make predictions.

Figure 8 compares the hierarchical to the base TDGP. All predictor signature tables use unlimited storage, and the best address-bit configuration (8 bits for commercial, 16 bits for scientific) is used for the base predictor and the high-bit table in the hierarchical design. The commercial workloads see coverage gains of 8%-14%, except for the static web servers, which were previously above 90% and gain 3%-4%. This significant performance increase stems from the flexibility offered by two signature tables. The zero-bit table learns quickly as program behavior changes over time, while the high-bit table captures stable but

complex behavior. Moreover, H-TDGP takes advantage of multiple views into the application’s history to overcome the detrimental effects of subtrace aliasing, which is common in the commercial workloads.

The increase in coverage arises from cooperation between the tables. Approximately 10% of all productions can be predicted by the zero-bit table, but not the 8-bit table, while the converse accounts for approximately 15%. This indicates that new signatures are encountered even after the workloads enter steady-state behavior, which is not surprising given the programming techniques used in commercial applications. Therefore, we cannot expect a TDGP design to improve over the baseline if it does not provide mechanisms for exploiting both signatures that need address disambiguation and those that do not.

Misprediction rates range from 9% to 38%, which will put significant strain on the memory consistency model to hide the resulting latency penalties. However, the percentage increase in mispredictions is small, and therefore likely tolerated by any system that can already handle the mispredictions incurred by a base TDGP.

The scientific applications exhibit no change in performance under the hierarchical predictor. As long as a single 16-bit signature table has sufficient capacity for all required signatures, the hierarchical approach cannot increase coverage. However, some signatures do not require address disambiguation, and therefore will be placed only in the zero-bit table, saving storage in the high-bit table for signatures that will benefit from address information. Moreover, this diminishes the need to scale the signature table with the size of the dataset being computed, enabling a particular H-TDGP configuration to be effective over a wider range of scientific workloads.

Table 3: Signature Reduction for Scientific Applications under H-TDGP

| Application | Reduction in Number of Signatures |
|--------------------|--|
| <i>barnes</i> | 15% |
| <i>em3d</i> | > 99% |
| <i>moldyn</i> | 65% |
| <i>ocean</i> | 50% |

Table 3 shows the reduction in the number of signatures created in the hierarchical versus the baseline predictor. As previously seen in Figure 4, *em3d* exhibits no sensitivity to TDGP's address-bit configuration, indicating that very few signatures are needed. H-TDGP detects this through its confidence mechanisms and generates the minimum number of signatures necessary (fewer than one hundred). The other scientific applications observe a significant reduction in signatures, but not so remarkable as with *em3d*, because they require some degree of address disambiguation.

6 Conclusion

In this thesis, we studied two predictors for memory coherence activity in distributed shared memory architectures. TDGP predicts the last store to a cache block prior to consumption by another node, and PCSP predicts the consumers of updated cache blocks. We evaluated these predictors on commercial workloads and determined that TDGP correctly predicts 51%-94% of productions, while PCSP is largely ineffective because of the inherent non-repetitiveness of memory access patterns in these applications. We proposed and evaluated a hierarchical TDGP design that exploits multiple underlying predictor tables and yields increased coverage of 63%-97%.

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] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzky. Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, May 2004.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.
- [13] 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.
- [14] 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.
- [15] S. Somogyi, T. F. Wenisch, N. Hardavellas, J. Kim, A. Ailamaki, and B. Falsafi. Memory coherence activity prediction in commercial workloads. In *Proceedings of the 3rd Workshop on Memory Performance Issues*, June 2004.

- [16] Standard Performance Evaluation Corporation. SPECjbb2000. <http://www.spec.org/jbb2000/>, 2000.
- [17] 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.
- [18] 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, Dec. 1991.