Adaptive Set Pinning: Managing Shared Caches in Chip Multiprocessors *

Shekhar Srikantaiah Mahmut Kandemir Mary Jane Irwin

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
{srikanta, kandemir, mji}@cse.psu.edu

Abstract

As part of the trend towards Chip Multiprocessors (CMPs) for the next leap in computing performance, many architectures have explored sharing the last level of cache among different processors for better performance–cost ratio and improved resource allocation. Shared cache management is a crucial CMP design aspect for the performance of the system. This paper first presents a new classification of cache misses – CII: Compulsory, Inter-processor and Intra-processor misses – for CMPs with shared caches to provide a better understanding of the interactions between memory transactions of different processors at the level of shared cache in a CMP. We then propose a novel approach, called set pinning, for eliminating inter-processor misses and reducing intra-processor misses in a shared cache. Furthermore, we show that an adaptive set pinning scheme improves over the benefits obtained by the set pinning scheme by significantly reducing the number of off–chip accesses. Extensive analysis of these approaches with SPEComp 2001 benchmarks is performed using a full system simulator. Our experiments indicate that the set pinning scheme achieves an average improvement of 22.18% in the L2 miss rate while the adaptive set pinning scheme reduces the miss rates by an average of 47.94% as compared to the traditional shared cache scheme. They also improve the performance by 7.24% and 17.88% respectively.

Categories and Subject Descriptors B.3.2 [*Memory Structures*]: Design Styles – Cache memory; C.4 [*Performance of Systems*]: Design Studies

General Terms Management, Design, Performance, Experimentation, Algorithms

Keywords Shared cache, Set pinning, CMP, Inter-processor, Intra-processor

* This research is supported in part by NSF grants CNS #0720645, CCF #0702519, a grant from Microsoft Corporation and support from the Gigascale Systems Research Focus Center, one of the five research centers funded under SRC's Focus Center Research Program.

ASPLOS'08, March 1–5, 2008, Seattle, Washington, USA.

1. Introduction

The most popular cache miss classification for uniprocessor architectures is the 3C miss classification (Compulsory, Capacity and Conflict misses) (9; 10; 11). According to this classification, compulsory misses are those misses caused by the first reference to a datum. Cache size and associativity make no difference in the number of compulsory misses. Prefetching can help here, as can larger cache block sizes (which are a form of prefetching). Capacity misses are those misses that occur regardless of associativity or block size, solely due to the finite size of the cache. Conflict misses are those misses that occur due to insufficient associativity (*i.e.*, they do not occur in a fully associative cache). This classification was further refined by Sugumar, et.al. (28; 29). They provide subclasses of conflict misses which are further classified as mapping misses, that are unavoidable given a particular degree of associativity and replacement misses, which are caused by a sub–optimal replacement policy. These classifications have helped researchers analyze the reasons for various classes of cache misses precisely. They have also aided the development of several performance optimizations which target reduction of specific kinds of cache misses and improve system performance in the case of uniprocessors (7; 19; 32; 33; 35). Dubois, et.al. (8) discovered a fourth category of misses in cache–coherent multiprocessors, called coherence misses. These are the misses that occur due to invalidation of cache entries shared between private caches in multi–processors. Coherence misses themselves are further classified as true–sharing misses and false–sharing misses based on whether the coherence miss was exactly due to the word written by one of the processors or due to independent words in the same cache block.

The latest versions of many architectures have chip multiprocessors (CMPs) with a shared L2/L3 cache (12; 15; 25). In these CMPs, the processors compete for the shared cache. In the context of CMPs with shared caches, the 3C classification of misses is inadequate to analyze the exact cause of misses and cannot model the contention that exists among the processors in accessing the shared cache. The ability to systematically characterize solutions to reduce misses in shared caches using the existing classifications is also limited. Coherence misses are used to model misses in multiprocessors with private caches. A new cache miss classification that accounts for the interactions between the transactions from different processors in a CMP is important in order to design schemes for effective shared cache management. We make the following important contributions in this paper:

- We provide a new classification of cache misses in the context of CMPs with shared cache. Our classification, *CII*, classifies the cache misses in a CMP with shared cache as Compulsory, Inter processor and Intra processor misses. Classifica-

Processor	Access	Action
P_1	Access location X	Cold Miss, **location X brought to cache**
P_1	Access location X	Cache Hit, No change
P_2	Access location X	Cache Hit, No change
	⋮	
P_1	Access location Y	**Replace location X**, Y brought to cache
P_1	Access location X	**Cache Miss**

(a) **Miss due to eviction of memory element by the same processor.**

Processor	Access	Action
P_1	Access location X	Cold Miss, **location X brought to cache**
P_1	Access location X	Cache Hit, No change
P_2	Access location X	Cache Hit, No change
	⋮	
P_2	Access location Y	**Replace location X**, Y brought to cache
P_1	Access location X	**Cache Miss**

(b) **Miss due to eviction of memory element by a different processor.**

Figure 1. Example transactions causing cache misses in a CMP with shared cache.

tion of non-compulsory cache misses in CII and 3C cache miss classification are orthogonal to each other. The CII classification helps us better understand the interactions between cache transactions of multiple processors in a CMP at the level of the shared cache.

- We propose a novel technique called *set pinning* which associates cache sets with owner processors (ownership in this paper refers to right of the processor to evict blocks within the set on a cache miss) and redirects blocks that would lead to inter-processor misses to a small *Processor Owned Private* (POP) cache. Each core has its own POP cache. Set pinning eliminates Inter-processor misses and reduces Intra-processor misses in shared caches. We also provide a quantitative analysis of the effect of set pinning on both inter-processor misses and intra-processor misses in a shared cache.
- As an evolutionary improvement over the set pinning approach, we propose a technique called *adaptive set pinning* which improves the benefits obtained by set pinning, by adaptively relinquishing ownership of pinned sets. The adaptive set pinning approach mitigates the effect of dominated ownership of sets by a few processors that is observed in the set pinning approach.
- We extensively evaluate the above approaches using a full system simulator and provide a characterization of the sensitivity of performance to various configuration parameters. In addition, we compare our approach to a victim cache (13) like configuration of POP cache. Adaptive set pinning achieves a performance improvement of 14.19% over a comparable victim cache configuration.

The remainder of the paper is organized as follows. Our CII classification is explained in Section 2. Section 3 and Section 4 elaborate on the set pinning and adaptive set pinning approaches. We present details about our simulation platform and benchmarks in Section 5 and experimental results in Section 6. Related work is described in Section 7, followed by our conclusions in Section 8.

2. A Taxonomy of Cache Misses in Chip Multiprocessors

The motivation for our classification springs from the example transactions depicted in Figure 1. Consider a CMP with two processors, P_1 and P_2 and a fully associative shared L2 cache. Exam-

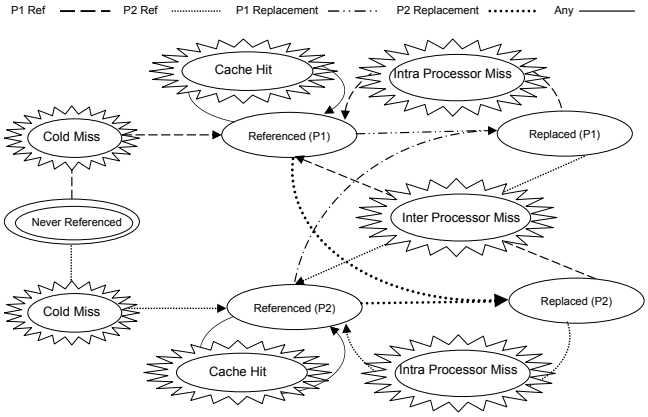


Figure 2. State diagram representing a memory element's life cycle in the shared cache. Non–compulsory misses are classified based on the processor responsible for evicting the referenced block. A non–compulsory miss is classified as ***Intra-processor miss*** if it was evicted by the same processor that brought it into the cache and ***Inter-processor miss*** if it was evicted by other processors.

ples (a) and (b) in Figure 1 show two possible types of transactions that could result in a miss in a shared cache. Example (a) depicts a traditional capacity miss where the same processor P_1 is responsible for both first reference and eviction of the memory element X. Example (b) also depicts a miss by P_1, that occurs to a memory element X. The difference here is that X was brought into the cache by an earlier reference by P_1, but was evicted only because of a reference to a different memory element Y that mapped to the same cache block as X by P_2.

Clearly, we would fail to understand the inherent differences in the cause for such misses, by classifying both of these misses as *"capacity misses"* (as in the 3C miss classification). The same is true with conflict misses. We classify the misses similar to that shown in (a) of Figure 1 as *Intra-processor misses* and ones similar to that shown in (b) as *Inter-processor misses*. Thus our classification, *CII*, classifies the cache misses in a CMP with a shared cache

into compulsory misses, intra-processor misses and inter-processor misses.

In order to present a more formal understanding of the CII classification, we can represent the life cycle of a memory element as shown in the state diagram in Figure 2. This diagram depicts the life cycle of a memory element in the shared cache during the execution of a program accessing it, assuming the program is executing on a dual core CMP. The same idea is easily extensible to any number of processors. As seen from Figure 2, the memory element under consideration is initially in the *Never Referenced* state. The first access by P_1 or P_2 causes a compulsory (cold) miss and the memory element enters the *Referenced* state for the first time in the life cycle. Any subsequent references (by any processor) to a memory element in the *Referenced* state leads to a cache hit. Further, a replacement of the cache block takes the memory element into the *Replaced* state. We tag the memory element with the id of the processor which replaced the element. For instance, a memory element which is evicted from the cache as a result of a reference from P_1 is in Replaced(P_1) state. It is evident that all non-compulsory cache misses to a memory element occur when it is in the *Replaced* state. Our classification of the non-compulsory misses is based on whether the cache miss is occurring because of the block being replaced (at an earlier point in time) by the same processor or a different processor. This is deciphered by comparing the processor facing the miss with the tag of the memory element in the *Replaced* state.

It is important to note that the classification of non-compulsory misses into intra-processor misses and inter-processor misses in the CII classification is orthogonal to the classification of the same as capacity and conflict misses. For instance, the examples discussed with reference to Figure 1, in case of a fully associative cache, represent (a) capacity miss that is also an intra-processor miss and (b) capacity miss that is also an inter-processor miss. Conflict misses can also be classified as intra-processor misses and inter-processor misses by the CII classification. Our CII classification is more expressive; and more importantly, it is able to model the interactions between transactions of multiple processors at the level of the shared cache.

We measured the distribution of various classes of misses in the CII classification. Figure 3 plots the distribution of compulsory, inter-processor and intra-processor misses in our base system configuration (see Section 5 for a detailed description of our baseline configuration). The black portion of the stacked bars represents the inter-processor misses, the spotted portion (in the middle) represents intra-processor misses and the striped portion represents the compulsory misses. On an average, 40.3% of the misses are inter-processor misses, 24.6% of the misses are intra-processor misses and the remaining 35.1% are compulsory misses.

Characterization of CII classification. We vary L2 cache size, L2 cache associativity and the number of processors individually from the baseline configuration in order to measure their impact on the distribution of various classes of CII cache misses. Graphs (a) and (b) of Figure 4 plot the normalized miss rates of various benchmarks by varying the size and associativity of the shared L2 cache. The miss rates are normalized to the total L2 cache miss rates of the respective benchmarks in the base configuration. Although the number of inter-processor and intra-processor misses in the L2 cache tend to decrease with increasing associativity and the size of the L2 cache, the contribution of inter-processor misses to the non–compulsory misses (as a percentage of non–compulsory misses) increases and that of intra-processor misses decreases as associativity and size increase. We also studied the impact of the number of processors on the CII classification. In order to have a fair comparison (the private L1 caches of a different number

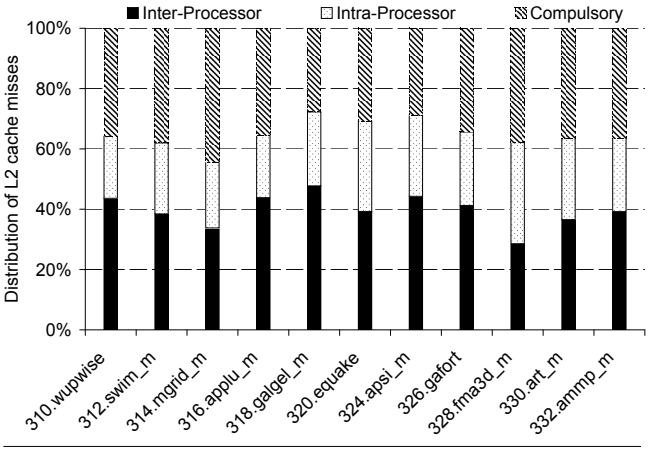


Figure 3. Distribution of CII misses in L2 cache misses for SPEComp benchmarks on an 8 processor CMP with 32KB, 4–way private instruction and data L1 caches and a 2MB 8–way associative shared unified L2 cache.

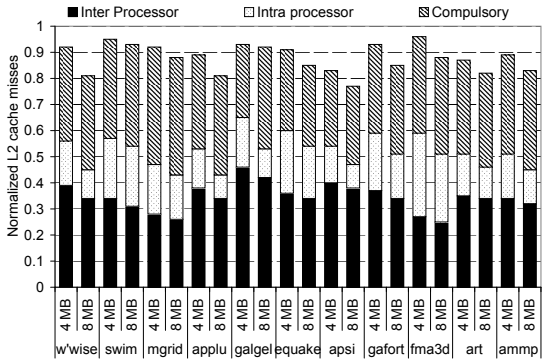
of processors lead to a different number of total L2 accesses) after varying the number of processors, we plot the variation in distribution of CII misses with the number of processors in Graph (c) of Figure 4. The contribution of inter-processor misses also increases with increasing number of processors. This is clearly due to the increased interaction between memory transactions of a higher number of processors in the shared L2 cache.

Reducing off–chip accesses is the key to a successful shared cache management scheme in a CMP with large shared L2/L3 cache (16). The effect of compulsory misses can be reduced by hiding their latency. This can be achieved by prefetching data into the cache before it is accessed. There have been many recent studies for reducing memory bandwidth and the number of off–chip accesses through hardware/software data prefetching (17; 24; 27; 34). The focus of this paper is on developing techniques to reduce inter-processor and intra-processor misses.

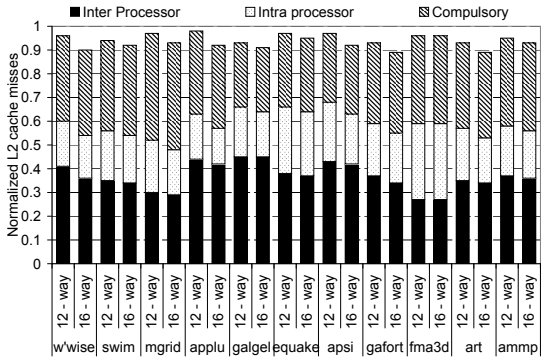
3. Set Pinning

CMPs with private caches for each processor have the advantage that there are no inter-processor misses. However, due to a much smaller fraction of aggregate L2 cache capacity being available to individual processors, there can be a significant increase in intra-processor misses (all cache misses in a private cache are intra-processor misses), compared to a shared L2 cache of a much larger (aggregate) capacity. Thus there is a motivation for shared caches in CMPs. Increasing the associativity of the shared L2 cache is another easy way of reducing intra-processor misses but, as seen from Figure 4, inter-processor misses then become the bottleneck as far as reducing non-compulsory misses is concerned. Higher associativity also results in higher power consumption, higher complexity and increased latency overheads. We propose a novel shared cache management scheme called set pinning to eliminate inter-processor misses and to reduce intra-processor misses without incurring the penalties of increasing associativity of the shared cache.

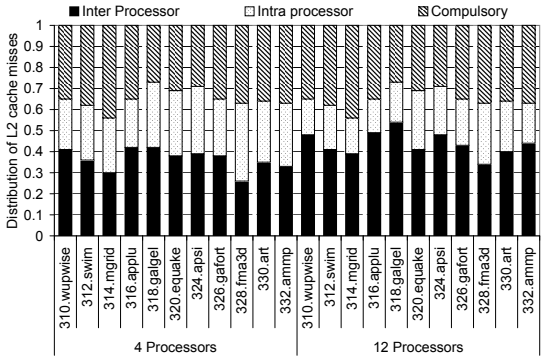
Set pinning is based on two crucial observations about the characteristics of non–compulsory misses in the shared cache. We measured the number of distinct memory addresses in the references that lead to inter-processor and intra-processor misses. The fraction of the number of references to distinct memory addresses resulting in inter-processor and intra-processor misses are plotted in Graph (a) of Figure 5. The low fraction of distinct memory ad-



(a) **Sensitivity to L2 cache size.**



(b) **Sensitivity to associativity.**



(c) **Sensitivity to number of processors.**

Figure 4. Sensitivity of distribution of compulsory, inter-processor and intra-processor misses in L2 cache to various parameters. All miss rates are normalized ***w.r.t*** the total L2 cache miss rates of respective benchmarks in our baseline configuration.

dresses leading to inter-processor misses indicates that most of the inter-processor misses happen due to a few *hot blocks* in the memory. We also measured the time interval in terms of the number of intervening references between successive references to these hot blocks and the result is plotted in Graph (b) of Figure 5. As seen from this graph, most of these hot blocks are accessed over and over again within 100 references 64.5% of the time on average. This indicates that the hot blocks are also frequently accessed. Set pinning is designed to exploit these two observations by disallowing the large number of references for the few *hot blocks* responsible for causing inter-processor misses, from evicting L2 cache blocks. Instead, these *hot blocks* are stored in very small regions of the L2 cache, confined to be written by individual processors, called *Processor Owned Private* (POP) caches.

Set pinning is a cache management scheme where every processor acquires *replacement ownership* of a certain number of sets in the shared cache. Only the processor that has replacement ownership of the set being accessed can replace entries in that set. Therefore, in set pinning, all references that could potentially cause an inter-processor miss, *i.e.*, all references from different processors, could never evict each other even if they index to the same set. The small number of such hot–blocks that would have been responsible for inter-processor misses are stored instead in the POP cache of the processor that first references them.

The block diagram of the proposed set pinning architecture for L2 cache is shown in Figure 6. As seen from the figure, we need a field in each cache set to store the identifier of the current owner processor of the set. This field is $\log n$ bits for n processors per cache set which corresponds to less than 0.6% overhead for our baseline configuration. The large shared L2 cache is organized into a large set pinned cache along with small POP caches, one for each processor. The set pinned L2 cache behavior is unchanged from the traditional shared cache behavior in case of a cache hit. Look up also happens in parallel in all the POP caches and on a cache miss in the set pinned L2 cache; if there is a hit in any of the POP caches, the cache request is satisfied from the POP cache.

3.1 Cache Miss Policy.

When there is no tag match in either the indexed set of the set pinned L2 cache or any of the POP caches, it is an L2 cache miss. There are three possible cases under which a cache miss can occur:

1. The indexed set in the L2 cache is not owned by any processor.
2. The indexed set in the L2 cache is owned by the processor responsible for the current reference.
3. The indexed set in the L2 cache is owned by a processor other than the one responsible for the current reference.

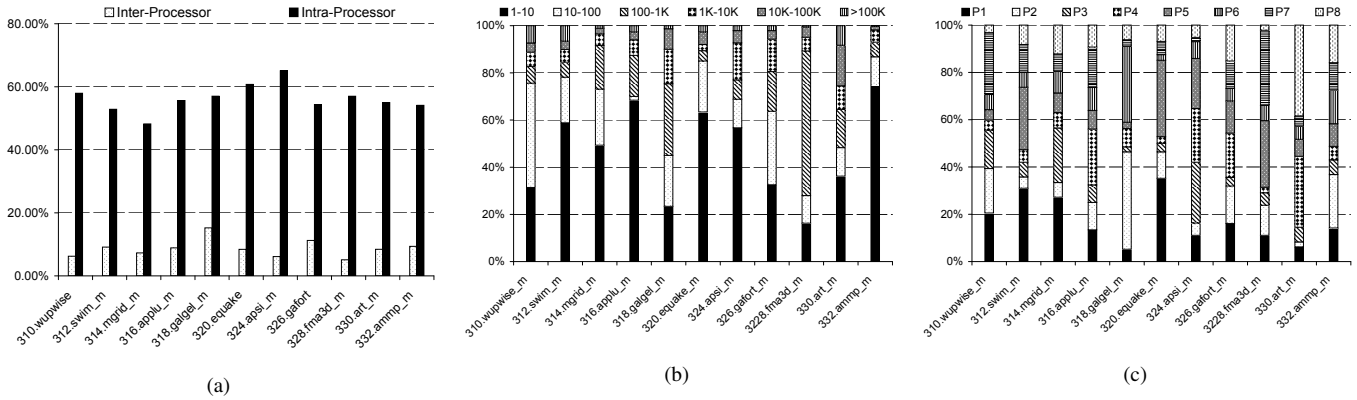


Figure 5. (a) Fraction of references to distinct memory addresses leading to intra-processor misses and inter-processor misses in the set pinning architecture, (b) Temporal locality of references leading to inter-processor misses, (c) Domination of ownership in set pinning.

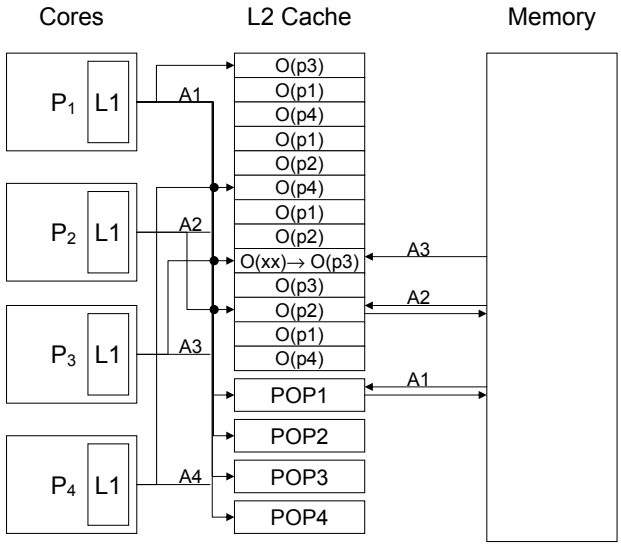


Figure 6. Block diagram explaining the working of our set pinning architecture.

In case 1, the indexed set is in its pristine state and is not owned by any processor. Therefore, in addition to bringing the referenced data from memory, the owner field also needs to be set. Consider the reference to A3 by P_3 in Figure 6. Since the address indexes to a block in a set owned by no processor (indicated by o(xx)), the ownership of the set is claimed by P_3 and the referenced data is brought into the set from memory.

In case 2, the indexed set is used as in the traditional cache schemes, *i.e.,* the data from memory evicts the least recently used cache block in that set. Intra-processor misses can be significantly reduced because of reduced number of references that are eligible to evict data (limited to references from the owner processor). The reference to A2 by P_2 in Figure 6 is an example of this case. Since the address indexes to a set owned by P_2, an LRU block is evicted from the indexed set and replaced by the referenced data.

In case 3, where the indexed set is owned by a processor different from the one making the reference, data cannot be evicted from the indexed set. This reduces the inter-processor misses which could only occur if the eviction had taken place. The POP cache owned by that processor is instead used to store the referenced data by evicting the LRU entry from that processor's POP cache. Considering Figure 6 once more, let us focus on A1 referenced by P_1 which indexes to a set owned by P_3. In this case, P_1 stores A1 in POP-1 cache, by evicting a block selected using LRU from the POP1 cache.

It is important to note that, in the set pinning architecture, there is no replication or migration of cache blocks between the set-pinned L2 cache and the POP caches. Every cache block is present in either the set pinned L2 cache or the POP cache of the processor which references it first. Also, the set pinning architecture does not impose any limitations on optimizing the L2 cache organization, *i.e.,* popular techniques such as multi–banking as well as migration and replication techniques within the L2 cache can be used to reduce costly off–chip accesses or power consumption. A detailed analysis of the impact of set pinning on inter-processor misses, intra-processor misses and overall performance is presented in Section 6.

4. Adaptive Set Pinning

The policy of allocating ownership of sets to processors that is described in the previous section is based on a first–come first–serve allocation policy. This simple policy could potentially result in an unfair division of the sets in the set pinned L2 cache. The issue of fairness in acquiring ownership is addressed by our next approach called adaptive set pinning described here in detail.

4.1 Dominated Ownership of Sets

The simple first–come first–serve policy favors the processors that first access the set. This can lead to acquirement of ownership of a large number of sets by the first few active processors. Such a domination in ownership of sets by a few processors leaves few sets for the other processors and hence over stresses the POP caches of those processors. In order to study the degree of such domination, we measured the percentage of sets owned by each processor at the end of execution (after they acquire ownership of maximal sets) with this simple first–come first–serve policy of allocating ownership. The results obtained are plotted in Graph (c) of Figure 5. As seen from this figure, a few processors dominate ownership of most sets. Consequentially, a fairer adaptive policy needs to be

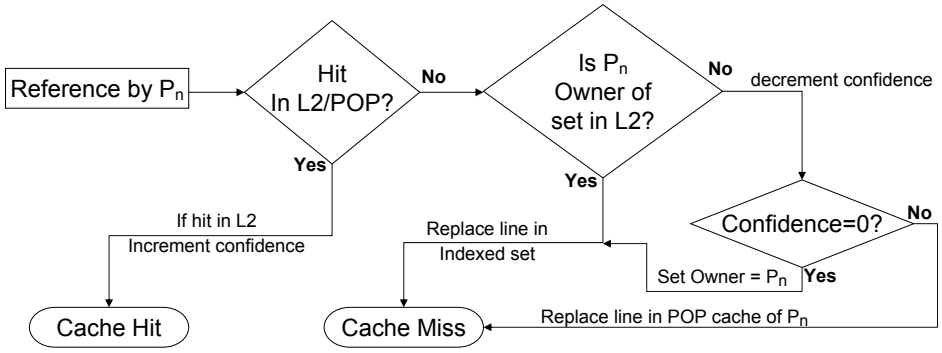


Figure 7. Flow chart explaining the logic of the adaptive set pinning scheme.

adopted by processors in acquiring ownership, so that they do not harm the interests of the performance of the system as a whole.

4.2 Relinquishing Ownership

The problem in domination of set ownership arises because the sets, once owned by a processor can never be owned by any other processor, irrespective of the activity level of that processor in the shared L2 cache. In the adaptive set pinning scheme, we overcome this limitation with a policy where processors dynamically relinquish ownership of sets.

Hardware Support. The relinquishing of sets by an owner processor is based on a *confidence counter* for each set, which indicates the confidence of the system that the current processor should be the owner of the set. The confidence counter is a saturating counter which is incremented on every cache hit occurring on the set. It is decremented for every reference by a processor, that (i) indexes into the set, (ii) is not the owner of the set, and (iii) is a cache miss (the reference is a cache miss in the indexed cache set as well as the POP caches). A cache miss due to a reference indexing into a set owned by the processor responsible for the reference does not change the value of the confidence counter.

When the confidence counter becomes zero, the processor identity bits are cleared and the set again becomes available to the next processor whose reference indexes into it. The initial value of the confidence counter is set to half of the saturating value. Note that any change in ownership of sets only means a change in the processor which can evict blocks from the cache set. A change in ownership does not invalidate any cache blocks. It also does not affect references from other processors to already existing cache blocks (unless they are evicted by the new owner).

The flow chart for the logic in the adaptive set pinning scheme is shown in Figure 7. Along with the processor identifier field for the set pinning architecture, the total additional hardware cost, assuming 4–bit confidence counters per set is about 1.36% of the L2 cache in our baseline configuration. 4 bit confidence counters were used after experimenting with a range of values from 2 to 16 and finding 4 bits was sufficient to account for the longest duration of ownership without frequent saturations.

5. Experimental Setup

In this section we describe our evaluation methodology. All our results are obtained with the baseline configuration system described below.

Base System Configuration. We evaluate both set pinning and adaptive set pinning on an eight processor CMP with processors based on the SPARC V9 architecture. The baseline configuration is given in Table 1. All configurations involving the evaluation of set pinning architecture have been evaluated with a reduced L2 cache capacity in order to match the total capacity in the schemes being compared.[1] We simulate the entire system using Simics full system simulator (18). A detailed analysis of the characteristics of CII misses by varying parameters such as cache size, associativity and number of processors was presented in Figure 4 and discussed in detail in Section 2.

Table 1. Experimental setup.

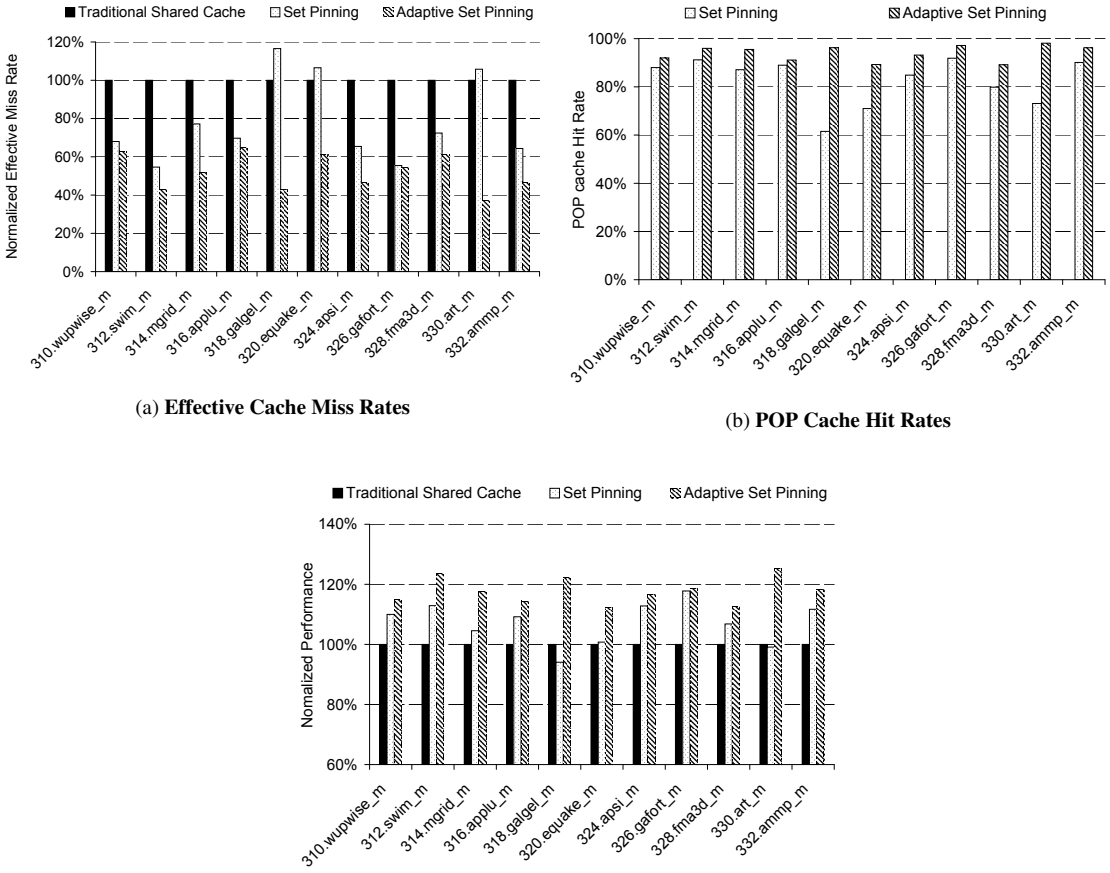
Processors	8 Processors with private L1 data and instruction caches, each processor is single threaded
Processor Model	Each processor is a 4–way fetch and issue in–order processor, 1GHz frequency
Private L1 D–Caches	Direct mapped, 32KB, 64 bytes block size, 3 cycle access latency
Private L1 I–Caches	Direct mapped, 32KB, 64 bytes block size, 3 cycle access latency
Shared L2 Cache	8–way set associative, 2MB (reduced to 1920 KB in presence of POP caches), 64 bytes block size, 15 cycle access latency
Memory	4GB, 200 cycle off–chip access latency
POP Cache	8–way set associative, 16KB per processor, 64 bytes block size

Table 2. Important characteristics of the SPEComp benchmark programs.

	Cache miss rates		
Benchmark	**L1**	**L2**	**Memory Footprint (MB)**
310.wupwise_m	1.6	13.2	1480
312.swim_m	8.1	25.4	1580
314.mgrid_m	11.3	18.2	450
316.applu_m	0.8	14.9	1510
318.galgel_m	1.1	11.3	370
320.equake_m	2.4	9.7	860
324.apsi_m	8.4	7.6	1650
326.gafort_m	6.8	13.5	1680
328.fma3d_m	4.1	8.3	1020
330.art_m	0.9	43.1	2760
332.ammp_m	0.6	5.9	160

Benchmarks. To quantitatively analyze the CII classification and to evaluate the effectiveness of set pinning and adaptive set pinning on CMPs, we used all the programs from the SPEComp 2001 benchmark suite (1). All the benchmark programs use the reference input set and are fast forwarded to the beginning of the main loops.

[1] Reduced to 1.92 MB to account for a total of 128 KB of POP cache capacity. The effect of hashing is considered but could be avoided in real cache systems that are designed with POP cache.



(a) **Effective Cache Miss Rates**

(b) **POP Cache Hit Rates**

(c) **Performance**

Figure 8. Evaluation of important metrics measuring the effectiveness of our schemes.

We warm up caches for approximately 1 billion instructions and collect statistics until the end of the loop iterations. The cache miss characteristics and memory footprints of the benchmark programs using our baseline configuration (without POP cache) are provided in Table 2.

6. Experimental Results

In this section, we present a detailed experimental study of the impact of set pinning and adaptive set pinning on inter-processor misses, intra-processor misses and the performance of the benchmark programs. We further show how our schemes perform with a varying number of processors. Finally, we compare our schemes with a victim cache (13) configuration of POP cache of similar size.

6.1 Cache Miss Rates

Both set pinning and adaptive set pinning do not influence the number of compulsory cache misses as explained earlier. Set pinning eliminates inter-processor misses, but they may manifest as a few additional intra-processor misses in the POP caches. Therefore, the effective number of misses in our schemes are the misses that occur in both the set pinned L2 cache and the POP caches. The effective L2 cache miss rate in the set pinning scheme and adaptive set pinning scheme is defined as follows:

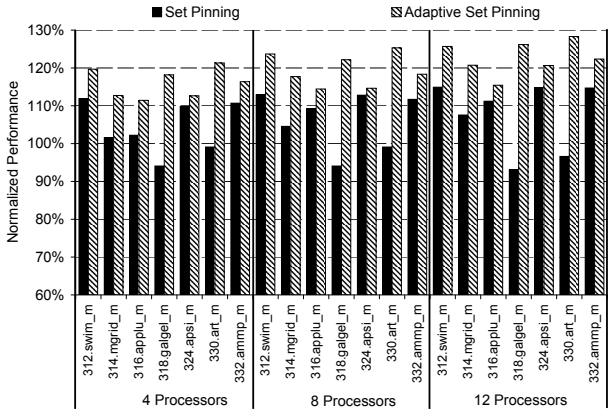
$$MR_{eff} = \frac{SetPinnedL2\ misses \bigcap POPcache\ misses}{Total\ L2\ accesses}$$

The effective L2 cache miss rates for set pinning and adaptive set pinning normalized with miss rates of the traditional shared cache scheme (base configuration) are plotted in graph (a) of Figure 8. In some benchmarks like 318.galgel_m and 330.art_m, the effective miss rate in the case of set pinning is higher than that of the traditional shared cache scheme. This happens due to dominated ownership of sets by a few processors in the set pinning scheme. Confidence counter based relinquishing of ownership in the adaptive set pinning scheme is effective in reducing the miss rates in both these cases. The set pinning scheme achieves an average improvement of 22.18% in the effective L2 miss rate, while the adaptive set pinning scheme reduces the miss rates by an average of 47.94%, as compared to the traditional shared cache scheme.

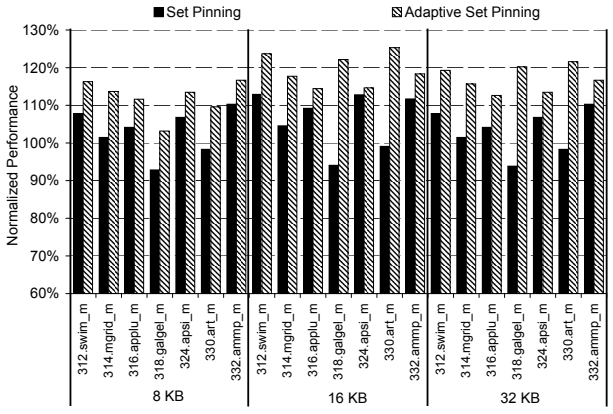
Another important metric that determines the performance of our schemes is the effective hit rate of the POP caches. We define this metric as:

$$POP\_hit\_rate = \frac{Total\ hits\ in\ POP\ caches}{Misses\ in\ SetPinnedL2}$$

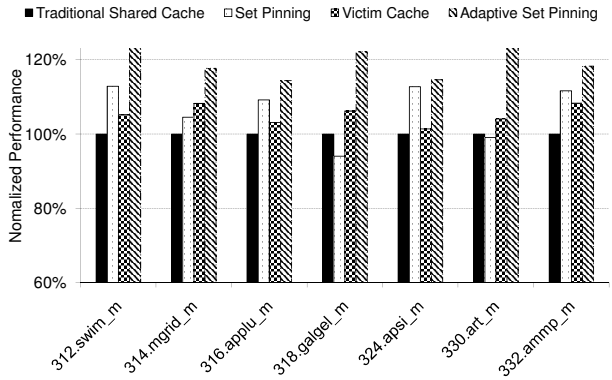
The effective hit rates in POP caches in set pining and adaptive set pinning are shown in graph (b) of Figure 8. The POP cache hit rates



(a) **Effective Cache Miss Rates**



(b) **POP Cache Hit Rates**



(c) **Performance**

Figure 9. (a) Sensitivity to number of processors. (b) Sensitivity to POP cache capacity. (c) Comparison to victim cache

are found to improve significantly with adaptive set pinning, with an average of 94.23% across all benchmarks, which is about 11.7% better than the hit average rate in the set pinning scheme.

6.2 Performance

Set pinning and adaptive set pinning significantly improve the performance of the workloads where they significantly reduce misses. Graph (c) of Figure 8 plots the speedups achieved by set pinning relative to traditional shared caches. Set pinning alone improves the performance by an average of 7.24%. The benchmarks influenced by dominated ownership of sets, like 318.galgel_m and 330.art_m, face a small hit on the performance (5.93% and 0.90% respectively) in case of the baseline set pinning approach. Adaptive set pinning significantly improves performance of all the benchmarks with speedups ranging from 12.17% to 25.31%, with an average speedup of 17.88% across all evaluated benchmarks.

6.3 Sensitivity Analysis

Number of processors. Graph (a) of Figure 9 plots the speedups obtained by set pinning and adaptive set pinning schemes relative to the traditional shared cache scheme with 4, 8 and 12 processors for a representative set of programs. The adaptive set pinning scheme clearly performs better with increasing number of processors. The adaptive scheme achieves an average of 3.3% and 6.7% additional speedups across benchmarks with 12 processors, as compared to 4 processors and 8 processors, respectively.

POP cache capacity. The sensitivity of performance in various schemes to the aggregate capacity of POP caches (and accompanying reduction in capacity of set pinned L2) is captured in Graph (b) of Figure 9. The values plotted represent the performance speedups of programs with set pinning and adaptive set pinning relative to the performance of the programs with traditional shared cache with baseline configuration (2 MB shared cache). Both 8KB and 32KB POP cache (per processor) result in smaller speedups compared to using the 16 KB per POP cache. POP cache miss rates increase with reduced POP cache capacity, leading to reduced speedups. An increase in POP cache size is also detrimental to the performance speedups. The 32KB POP cache per processor achieves speedups close to the 16KB configurations (smaller by 2%). The major factor for the reduction in performance with increased POP cache capacity is the reduction in the available set pinned L2 capacity.

6.4 Comparison with Victim Cache

In this section, we briefly compare the performance of our set pinning schemes with POP caches to using a victim cache. Many pro-

posed cache partitioning schemes are not as dynamic as adaptive set pinning and hence not directly comparable. A Victim cache (13) is an approach used to reduce capacity and conflict misses by storing the victims of the replacement policy in a small buffer. The subtle, but important difference between our approach and the victim cache is that the victim cache caches the victims after they are evicted by the replacement policy, while we modify the replacement policy such that the cache blocks that would victimize other processor's elements (causing inter-processor misses) are themselves cached in the POP caches. In order to create as realistic a match as possible and to have a fair comparison in terms of the total cache capacity as well as the parameters involved like set associativity, we compare with identical aggregate POP cache capacity and victim cache capacity of 128 KB. We also have 64 way associativity for the victim cache. This is to match the effective associativity of the 8 POP caches for the 8 different processors. Graph (c) of Figure 9 plots the speedups obtained by our set pinning schemes and victim cache scheme relative to the performance of traditional shared cache scheme. The set pinning scheme performs marginally better than the victim cache scheme (better by 1.07% on average), while the adaptive set pinning scheme achieves significant speedups over the victim cache scheme (14.19% on average). This significant improvement in the speedup can be explained based on the following intuitive reasoning. The victim cache is responsible for caching both inter-processor and intra-processor misses, while the POP caches handle only those references which would cause inter-processor misses. As seen earlier, from graph (a) of Figure 5, the number of distinct memory elements that face an intra-processor miss is far higher than those facing an inter-processor miss. Consequently, the victim cache is responsible for caching many more entries than a corresponding POP cache configuration. This results in an average victim cache miss rate of 19.1% across the benchmarks, which is 1.6% and 13.3% more than in case of set pinning and adaptive set pinning, respectively. Comprehensive comparisons with other cache miss reduction schemes are not presented here due to space constraints.

7. Related Work

In this section, we briefly review the most directly related studies and describe how our contributions differ from them or supplement them.

Shared Cache Management in CMPs. Recently, many researchers have explored hybrid CMP cache designs that attempt to achieve the low access latency advantages of private L2 caches and the low off-chip miss rates of shared L2 caches (3; 5; 6; 36). Static and dynamic shared cache partitioning schemes have been explored in order to provide for isolation (4; 14; 21; 22; 30). Researchers have also explored in depth similar problems in distributed VOD systems (26).

Zhang and Asanovic (36) present victim replication which is a variant of the shared scheme that attempts to keep copies of local primary cache victims within the local L2 cache slice but allows multiple copies of a cache block to co-exist in different L2 slices of the shared L2 cache.

Chang and Sohi (5) propose cooperative caching which uses cooperation mechanisms between private L2 caches to reduce costly off–chip accesses. One of the mechanisms used by cooperative caching is to put a locally evicted block in another on-chip L2 cache that may have spare capacity, rather than evict it from the on-chip hierarchy entirely. In set–pinning and adaptive set–pinning, we instead put the blocks which would cause an existing block to be evicted in the POP cache to reduce off–chip accesses thereby eliminating inter-processor misses and reducing intra-processor misses in CMPs.

Beckmann and Wood (2) have proposed Adaptive Selective Replication (ASR), a mechanism that dynamically monitors workload behavior to control replication. ASR replicates cache blocks when it estimates that the benefit of replication in terms of lower L2 hit latency exceeds the cost due to increased L2 misses. Adaptive Selective Replication can be used along with our schemes to improve the L2 hit latency.

Petoumenos (14) has studied modeling of cache sharing or partitioning similar to the cache decay model. Their model known as StatShare directly describes capacity misses and can estimate cold misses but fails to take into account conflict misses. Our model is complementary to this and most importantly, set pinning and adaptive set pinning help in reducing conflicts among multiple processors in the cache.

CMP shared cache management schemes at the software level have also been researched in the recent past. Rafique (23) proposes an OS scheme that consists of a hardware cache quota management mechanism, an OS interface and a set of OS level quota orchestration policies for greater flexibility in policies. Tam (31) proposes a software mechanism in the OS that allows for partitioning of the shared L2 cache by guiding the allocation of physical pages. Any such software scheme provides higher flexibility at the cost of restricted applicability as compared to a hardware scheme.

Cache Miss Classifications. A detailed description of related work in classifying cache misses was presented in Section 1. To the best of our knowledge, ours is the first work which systematically classifies the cache misses in CMPs with shared cache based on the interaction between memory transactions from different processors.

Conflict Miss Reduction. Reduction of cache conflict misses in private caches of uniprocessors has been a hot topic of research for more than two decades now and there have been several important works that address this problem for uniprocessors in both hardware and software (7; 13; 20; 32; 33; 35). Collins and Tullsen (7) suggested the use of a hardware miss classification table that enables the processor or memory controller to identify each cache miss as either a conflict miss or a capacity (non-conflict) miss. The miss classification table works by storing part of the tag of the most recently evicted block of a cache set. If the next miss to that cache set has a matching tag, it is identified as a conflict miss. Their work also highlights the significance of classification of misses to optimizations specifically targeting them. Our work does not directly address conflict misses (or capacity misses), but it is the first to view these misses from a different perspective in order to understand the inherent reason behind such misses in a CMP with a shared cache.

8. Conclusions and Future Work

The proposed CII scheme of classification of shared L2 cache misses helps us to better understand the interactions between cache transactions of multiple processors in a CMP. Architectural techniques exploiting this understanding to improve the performance of the system are also proposed. The set pinning and adaptive set pinning schemes are proposed with improvement of performance as the primary objective. As a result, a few components like the parallel tag comparators in POP caches and the confidence counters in adaptive set pinning scheme have not been optimized for power consumption. Work in this direction is underway. Evaluation of performance benefits obtained for multiprogrammed workloads with our schemes is also being conducted. Exploring the possibility of a collection of processors owning a set (instead of just one) is another interesting problem being explored. The adaptive set pinning scheme is shown to be effective in improving the performance by reducing costly off–chip accesses. The set pinning scheme achieves

an average improvement of 22.18% in the miss rate while the adaptive set pinning scheme reduces the miss rates by an average of 47.94% as compared to the traditional shared cache scheme leading to a performance improvement of 17.88% on an average.

References

[1] V. Aslot, M. J. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones, and B. Parady. Specomp: A new benchmark suite for measuring parallel computer performance. In *Proc. of the International Workshop on OpenMP Applications and Tools*, pages 1–10, West Lafayette, 2001.

[2] B. M. Beckmann and D. A. Wood. ASR: Adaptive Selective Replication for CMP Caches. In *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Orlando, 2006.

[3] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *Proc. of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 319–330, Portland, 2004.

[4] J. Chang and G. S. Sohi. Cooperative Cache Partitioning for Chip Multiprocessors. In *Proc. of the 21st ACM International Conference on Supercomputing*, Seattle, 2007.

[5] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *Proc. of the International Symposium on Computer Architecture*, Boston, 2006.

[6] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Optimizing replication, communication, and capacity allocation in cmps. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, pages 357–368, Madison, 2005.

[7] J. D. Collins and D. M. Tullsen. Runtime identification of cache conflict misses: The adaptive miss buffer. *ACM Trans. Comput. Syst.*, 19(4):413–439, 2001.

[8] M. Dubois, J. Skeppstedt, L. Ricciulli, K. Ramamurthy, and P. Stenstrom. The detection and elimination of useless misses in multiprocessors. In *Proc. of the 20th Annual International Symposium on Computer Architecture*, pages 88–97, San Diego, 1993.

[9] J. L. Hennessy and D. A. Patterson. *Computer Architecture : A Quantitative Approach; second edition*. Morgan Kaufmann, 1996. HEN j2 96:1 1.Ex.

[10] M. D. Hill. *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, EECS Department, University of California, Berkeley, 1987.

[11] M. D. Hill and A. J. Smith. Evaluating associativity in cpu caches. *IEEE Transactions on Computers*, 38(12):1612–1629, 1989.

[12] L. Hsu, R. Iyer, S. Makineni, S. Reinhardt, and D. Newell. Exploring the cache design space for large scale cmps. *SIGARCH Computer Architecture News*, 33(4):24–33, 2005.

[13] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of the 17th Annual International Symposium on Computer Architecture*, Seattle, 1990.

[14] S. Kim, D. Chandra, and Y. Solihin. Fair cache sharing and partitioning in a chip multiprocessor architecture. In *Proc. of the 13th International Conference on Parallel Architectures and Compilation Techniques*, Paris, 2004.

[15] P. Kongetira, K. Aingaran, and K. Olukotun. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 25(2):21–29, 2005.

[16] C. Liu, A. Sivasubramaniam, and M. Kandemir. Organizing the last line of defense before hitting the memory wall for cmps. In *Proc. of International Symposium on High Performance Computer Architecture*, Madrid, 2004.

[17] J. Lu, A. Das, W.-C. Hsu, K. Nguyen, and S. G. Abraham. Dynamic helper threaded prefetching on the sun ultrasparc cmp processor. In *Proc. of the International Symposium on Microarchitecture*, Barcelona, 2005.

[18] 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. *Computer*, 35(2):50–58, 2002.

[19] G. Memik, G. Reinman, and W. H. Mangione-Smith. Just say no: Benefits of early cache miss determination. In *Proc. of the 9th International Symposium on High-Performance Computer Architecture*, Anaheim, February 2003.

[20] G. Memik, G. Reinman, and W. H. Mangione-Smith. Reducing energy and delay using efficient victim caches. In *Proc. of the 2003 International Symposium on Low Power Electronics and Design*, Seoul, 2003.

[21] P. Petoumenos, G. Keramidas, H. Zeffer, S. Kaxiras, and E. Hagersten. Modeling cache sharing on chip multiprocessor architectures. In *Proc. of the IEEE International Symposium on Workload Characterization*, 2006.

[22] M. K. Qureshi and Y. N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, Orlando, 2006.

[23] N. Rafique, W.-T. Lim, and M. Thottethodi. Architectural support for operating system-driven cmp cache management. In *Proc. of the 15th International Conference on Parallel architectures and Compilation Techniques*, Seattle, 2006.

[24] X. Shi, Z. Yang, J. Peir, L. Peng, Y.-K. Chen, Lee, and Liang. Coterminous locality and coterminous group data prefetching on chip-multiprocessors. In *Proc. of the 20th International Parallel and Distributed Processing Symposium*, Rhodes Island, 2006.

[25] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. Power5 system microarchitecture. *IBM J. Res. Dev.*, 49(4/5):505–521, 2005.

[26] B. Sonah and M. R. Ito. Modeling rate-based dynamic cache sharing for distributed vod systems. In *Proc. of the The International Conference on Information Technology: Coding and Computing*, page 489, 2000.

[27] L. Spracklen, Y. Chou, and S. G. Abraham. Effective instruction prefetching in chip multiprocessors for modern commercial applications. In *Proc. of International Symposium on High-Performance Computer Architecture*, San Francisco, February 2005.

[28] R. A. Sugumar. *Multi Configuration Simulation Algorithms for the Evaluation of Computer Architecture Designs*. PhD thesis, UMich, 1993.

[29] R. A. Sugumar and S. G. Abraham. Efficient simulation of caches under optimal replacement with application to miss characterization. In *ACM SIGMETRICS Conference on Measurment and Modeling of Computer Systems*, Santa Clara, 1993.

[30] G. E. Suh, L. Rudolph, and S. Devadas Dynamic Partitioning of Shared Cache Memory. *J. Supercomput.* 28, 1, Apr. 2004, 7-26.

[31] D. Tam, R. Azimi, L. Soares, and M. Stumm. Managing shared l2 caches on multicore systems in software. In *Proc. of the Workshop on the Interaction between Operating Systems and Computer Architecture*, San Diego, 2007.

[32] N. Topham, A. Gonzalez, and J. Gonzalez. The design and performance of a conflict-avoiding cache. In *Proc. of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, pages 71–80, Research Triangle Park, 1997.

[33] H. Vandierendonck, P. Manet, and J.-D. Legat. Application-specific reconfigurable xor-indexing to eliminate cache conflict misses. In *Proc. of the Conference on Design, Automation and Test in Europe*, pages 357–362, Munich, 2006.

[34] D. Wood and A. Alameldeen. Interactions between compression and prefetching in chip multiprocessors. In *Proc. of the 13th International Symposium on High-Performance Computer Architecture*, Phoenix, 2007.

[35] C. Zhang. Balanced cache: Reducing conflict misses of direct-mapped caches. In *Proc. of International Symposium on Computer Architecture*, Boston, 2006.

[36] M. Zhang and K. Asanovic. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Proc. of the 32nd Annual International Symposium on Computer Architecture*, Madison, 2005.