A Dynamic Pressure-Aware Associative Placement Strategy for Large Scale Chip Multiprocessors

Mohammad Hammoud, Sangyeun Cho, and Rami G. Melhem
Department of Computer Science, University of Pittsburgh
{mhh,cho,melhem}@cs.pitt.edu

Abstract—This paper describes dynamic pressure-aware associative placement (DPAP), a novel distributed cache management scheme for large-scale chip multiprocessors. Our work is motivated by the large non-uniform distribution of memory accesses across cache sets in different L2 banks. DPAP decouples the physical locations of cache blocks from their addresses for the sake of reducing misses caused by destructive interferences. Temporal pressure at the on-chip last-level cache, is continuously collected at a group (comprised of local cache sets) granularity, and periodically recorded at the memory controller(s) to guide the placement process. An incoming block is consequently placed at a cache group that exhibits the minimum pressure. Simulation results using a full-system simulator demonstrate that DPAP outperforms the baseline shared NUCA scheme by an average of 8.3% and by as much as 18.9% for the benchmark programs we examined. Furthermore, evaluations showed that DPAP outperforms related cache designs.

Index Terms—Chip Multiprocessors, Associative Placement, Pressure-Aware Placement, Aggregate Cache Sets, Local Cache Sets.

1 Introduction and Motivation

As large uniprocessors are no longer scaling in performance, chip multiprocessors (CMPs) have become the trend in computer architecture. CMPs can easily spread multiple threads of execution across various cores. Besides, CMPs scale across generations of silicon process simply by stamping down copies of the hard-to-design cores on successive chip generations [9]. One of the key challenges to obtaining high performance from CMPs is organizing and managing the limited on-chip cache resources (typically the L2 cache) shared among co-scheduled threads/processes.

Tiled chip multiprocessor (CMP) architectures have recently been advocated as a scalable processor design approach [4], [14]. They replicate identical building blocks (tiles) and connect them with a switched network on-chip (NoC) [14]. A tile typically incorporates private L1 caches and an L2 cache bank. L2 cache banks are accordingly physically distributed over the processor chip. A conventional practice, referred to as the shared scheme, logically shares these physically distributed cache banks. On-chip access latencies differ depending on the distances between requester cores and target banks creating a Non Uniform Cache Architecture (NUCA) [7].

Recent research work on CMP cache management has recognized the importance of the NUCA shared design [3], [4]. Besides, many of today's multi-core processors, the Intel Core^{TM}2 Duo processor family [12], Sun Niagara [8], and IBM Power5 [16], have featured shared caches. A shared organization, however, suffers from an interference problem. A defectively behaving application can evict useful L2 cache content belonging to other co-scheduled programs. As such, a program that exposes temporal locality can experience frequent cache misses caused by interferences. We observe that 69.5% of misses on a 16-way tiled shared CMP platform are inter-processor (a line being replaced at an earlier time by a different processor).[1]

We primarily correlate destructive interferences problem to the *root* of CMP cache management, the cache placement algorithm.

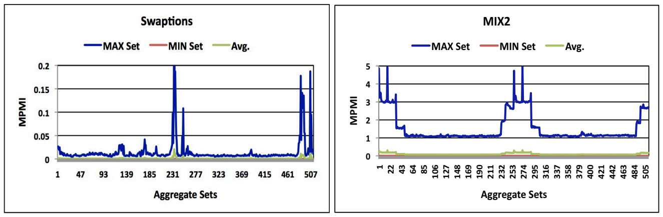

Fig. 1. **Number of misses per 1 million instructions (MPMI) experienced by two local cache sets (the ones that experience the max and the min misses) at different aggregate sets for two benchmarks, Swaptions and MIX2.**

Fig. 1 demonstrates the number of misses per 1 million instructions experienced by cache sets across L2 cache banks (or aggregate sets) for two benchmarks, Swaptions and MIX2 (see Section 4.1 for experimental details). We define an *aggregate set* with index i as the union of sets with index i across L2 cache banks. More formally, an $aggregate\ set_i = \bigcup_{k=1}^{n} set_{ki}$ where set_{ki} is the set with index i at bank k. We refer to each set_{ki} as a *local* set. We assume a 16-way tiled CMP platform with physically distributed, logically shared L2 banks. We only show results for two local sets that exhibit the maximum and the minimum misses, in addition to the average misses, per each aggregate set. Clearly, we can see that memory accesses across aggregate sets are asymmetric. A placement strategy aware of the current pressures at banks can reduce the workload imbalance among aggregate sets by preventing placing an incoming cache block at an exceedingly pressured local set. This can potentially minimize interference misses and maximize system performance.

Traditionally, cache blocks are stored at cache locations solely based on their physical addresses. This makes the placement process unaware of the disparity in the hotness of the shared cache sets. In this work, we explain the importance of incorporating pressure-aware associative placement strategies to improve CMP system performance. We propose dynamic pressure-aware associative placement (DPAP), a novel mechanism that involves a low-hardware overhead framework to monitor the L2 cache banks at a *group* (comprised of local cache sets) granularity and record pressure information at an array embedded within the memory controller. The collected pressure

Manuscript submitted: 18-Apr-2010. Manuscript accepted: 14-May-2010. Final manuscript received: 18-May-2010.
This work was supported in part by NSF grant CCF-0952273.

[1] Section 4.1 describes the adopted CMP platform, the experimental parameters, and the benchmark programs we examined.

Posted to the IEEE & CSDL on 5/25/2010
DOI 10.1109/L-CA.2010.7
1556-6056/10/$26.00 © 2010 Published by the IEEE Computer Society

information is utilized to guide the placement process. Upon fetching a block, *B*, with index *i* from the main memory, DPAP looks up the pressure array at the memory controller, identifies the bank with the group that exhibits the minimum pressure, and places *B* at the set with index *i* in that bank. As DPAP can place *B* at any local set with index *i* in any of the L2 banks independent of *B*'s address, we say that DPAP effectively provides a cache associativity that equates the aggregate associativity of the L2 banks. We generally refer to a placement process that exploits the aggregate associativity of the L2 cache banks as an *associative placement* strategy.

Major contributions of our work are as follows:

- We propose a practical pressure-aware associative placement mechanism that provides robust performance for distributed shared caches.
- We evaluate DPAP using a full system simulator and find that it successfully reduces cache misses of a shared CMP design by an average of 24.1%.
- We compare DPAP to various related schemes including victim caching (VC) [6], variable-way set associative cache (V-WAY) [11], dynamic set balancing cache (DSBC) [13], and cooperative caching (CC) [2].

The rest of the paper is organized as follows. Section 2 presents some recent related work. DPAP mechanism is detailed in Section 3. In Section 4 we evaluate DPAP. Conclusion and future directions are given in Section 5.

2 Related Work

Much work has been done to effectively minimize interference misses in conventional cache designs. It is, in fact, quite impossible to do justice to this large body of work in this short article. As such, we briefly describe some of the most closely related prior work.

A recent study, namely Dynamic Set Balancing Cache (DSBC) [13], suggests mitigating the large asymmetry in cache sets' usages via associating every two cache sets in a single cache (i.e., local sets), making the capacity of an underutilized set available for a pressured one. DSBC is suggested for single-core architectures. However, DSBC is directly extensible to CMPs. Nonetheless, DSBC alleviates the workload imbalance across only local cache sets. In contrast, DPAP attempts to minimize interference misses via reducing the uneven memory accesses across aggregate sets. In this work we show the potential of aggregate versus local set balancing. We compare DPAP and DSBC in Section 4.

Variable-Way Set Associative Cache (V-WAY) [11] increases the number of tag-store entries relative to the number of data lines to reduce interference misses. For the data-store, V-WAY promotes a global frequency based replacement policy. V-WAY is directly extensible to CMPs but is also essentially deemed as a local set balancing scheme. Section 4 compares V-WAY against DPAP.

In the context of CMPs, Adaptive Set Pinning (ASP) [17] associates processors to cache sets and solely grants them permissions to evict blocks from their sets on cache misses. Cooperative Caching (CC) [2] creates a globally managed shared aggregate on-chip cache on a private cache organization to reduce misses. In Section 4 we compare CC versus DPAP.

Both ASP and CC advocate CMP cache management at block granularity. At page granularity, Sherwood et al. [15] proposed a software page placement algorithm that performs coloring of virtual pages using profiles at compile time. Hardavellas et al. [4] proposed R-NUCA that relies on OS to classify cache accesses onto either private, shared, or instructions. In comparison to these page granular schemes, DPAP employs a block-grain placement strategy without any OS involvement. Hence, DPAP provides a transparent solution.

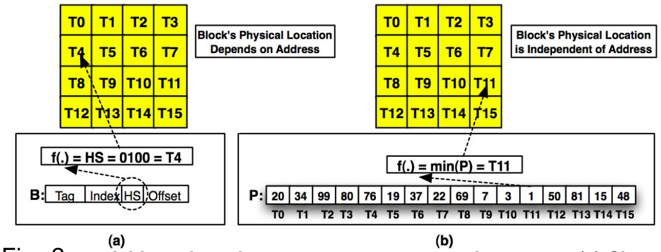
T0
T1
T2
T3
T4
T5
T6
T7
T8
T9
T10
T11
T12
T13
T14
T15
Block's Physical Location Depends on Address
f(.) = HS = 0100 = T4
B:
Tag
Index
HS
Offset
(a)
Block's Physical Location is Independent of Address
f(.) = min(P) = T11
P:
20
34
99
80
76
19
37
22
69
7
3
1
50
81
15
48
T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 T10 T11 T12 T13 T14 T15
(b)

Fig. 2. **Address-based versus pressure-aware placements. (a) Shared scheme strategy. (b) Pressure-aware associative strategy. (*f(.)* denotes the placement function, HS is the home select bits of block B, and P is the pressure array).**

3 The Proposed Mechanism

We propose a dynamic pressure-aware associative placement (DPAP) strategy that maps cache blocks to the L2 cache space depending on the observed pressures at the L2 cache banks. DPAP suggests that the pressure at each L2 bank is collected at run time, stored, and utilized to guide the placement process. Specifically, a pressure array is maintained at the memory controller(s) of the CMP system to store the collected pressures. On a miss to L2, the main memory is accessed and the pressure array is probed. The bank that exhibits the minimum pressure is selected to host the fetched cache block.

We gather pressures from individual sets or *groups* of sets at L2 banks. A cache bank can be divided into a number of groups. We denote a group size as the number of local sets (sets on the same bank) that a group includes. The dimension of the pressure array (rows vs. columns) at the memory controller(s) changes depending on the number of groups (*n*-group) per bank and the number of banks (*p*-bank). With *n*-group and *p*-bank the pressure array would consist of *n* rows and *p* columns. Therefore, a 1-group (i.e., bank) granularity indicates a linear pressure array and can be probed simply by using the IDs of the tiles. However, with finer granularities we need to select the row first (denoting the group number of an incoming cache block K) and then the column (denoting the bank that exhibits the minimum pressure for the selected group). The group number (GN) of a block, K, can be simply determined by dividing the index of K by the group size.

Fig. 2 demonstrates a descriptive comparison between the placement strategies of the nominal shared NUCA design and our proposed scheme. For simplicity we assume a linear pressure array. By using the shared scheme's placement strategy, a subset of bits, referred to as the home select (HS) bits, from the physical address of a requested block, B, is utilized to map B to its static home tile (SHT). Assuming the HS bits of B are 0100, B is accordingly placed at tile T4. Alternatively, by using our pressure-aware associative placement strategy, the pressure array at the memory controller is inspected before B is mapped to L2. The pressure array indicates that slot T11 has the minimum pressure, thus selected.

Typically, the pressure at an L2 bank can be measured in terms of cache misses or hits. However, it is not possible to measure cache misses in a meaningful way at L2 banks when a pressure-aware associative placement strategy is employed. Unlike an address-based placement strategy, on an L1 miss to a block B, there is no address that dictates the bank responsible for caching B. Besides, B might map to any bank (versus mapping only to the SHT on the nominal shared). As such, a reported L2 miss can't be correlated to any specific L2 bank but rather to the whole L2 cache space. Hence, we don't use misses to represent pressures at L2 banks but rather hits. More specifically, we quantify pressure as the number of lines that yield cache hits during a time interval, referred to as an *epoch*,

TABLE 1
System parameters

COMPONENT	PARAMETER
Cache Line Size	64 B
L1 I/D- Sizes/Ways/Latency *L1 Replacement Policy*	32KB/2-way/1 cycle LRU
L2 Cache Size/Ways/Latency *L2 Replacement Policy*	512KB per L2 bank/16-way/12 cycles LRU
Latency Per NoC Hop	3 cycles
Memory Latency	320 cycles

and designate that as *temporal pressure*.

DPAP doesn't rely on prior knowledge of the program but on hardware counters. A saturating counter per group can be installed at each tile to count the number of successful accesses to that group during an epoch. At the end of every epoch the values of the counters are copied from the local tiles to the pressure array at the memory controller(s). Besides, in order to allow DPAP to adapt to phase changes of applications, at the copy time we keep only 0.25 of the last epoch's pressure values (by shifting each value 2 bits to the right) and add to them the newly collected ones.

Finally, by having a pressure-aware associative placement algorithm, a location strategy capable of rapidly locating cache blocks at the L2 cache space is required. Clearly, the HS bits of a requested block can't be used anymore to locate the block. In this case, many strategies can be incorporated. First, a broadcast-based policy can easily fulfill the objective but might heavily burden the NoC. Second, a directory (either centralized or distributed) can be maintained and pointers can be kept to point to the current locations of blocks (as in [21]). This incurs, however, 3-way cache-to-cache transfers. A third option resolves the problem without broadcasting and with minimal 3-way communications and is referred to as cache-the-cache-tag (CTCT) [3] location policy.

DPAP adopts CTCT to achieve fast location of L2 cache blocks. Upon placing a cache block, B, at an L2 bank using DPAP, CTCT stores two corresponding tracking entries in special location tables (LT) at the requesting and the static home tiles of B. Subsequently, when the requesting core requests B and misses at L1, its LT table is looked up and if a hit is obtained, B is located directly at the L2 bank designated by the matched tracking entry at LT. Furthermore, if any other sharer core requests B, the SHT of B can be always approached and its LT table can be looked up to locate B at its current L2 bank. If no matching entry is found in SHT's LT table, an L2 miss is reported and the request is satisfied from the main memory. CTCT suggests that a tracking entry encompasses the tag of the related block (typically 22 bits), a bit vector to keep related tracking entries coherent (16 bits for a 16-tile CMP model), and an ID that points to the tile that is currently hosting the block (4 bits for 16 tiles).

4 QUANTITATIVE EVALUATION

4.1 Methodology

We present our results based on detailed full-system simulation using Virtutech's Simics 3.0.29 [19]. We use our own CMP cache modules fully developed in-house. We implement the XY-routing algorithm and accurately model congestion for both coherence and data messages. A tiled CMP architecture comprised of 16 UltraSPARC-III Cu processors is simulated running with Solaris 10 OS. Each processor uses an in-order core model with an issue width of 2 and a clock frequency of 1.4 GHz. The tiles are organized as a 4×4 grid connected by a 2D mesh NoC. Each tile encompasses a switch, 32KB I/D L1 caches, and a 512KB L2 cache bank. A distributed MESI-based directory protocol is employed. We adopt an epoch of 20 million instructions (see Section 3) and keep only 0.25 of the pressure values after every epoch. Besides, we consider a location table (LT) of 16K entries per tile required by the CTCT location policy [3]. Each access to an LT table requires 1.35ns estimated using CACTI v5.3 [5]. To justify the overhead incurred by CTCT we offer a study in the next subsection where we increase the cache size and associativity of the baseline shared CMP scheme. Table 1 shows our configuration's experimental parameters.

TABLE 2
Benchmark programs

NAME	INPUT
SPECJbb	Java HotSpot (TM) server VM v 1.5, 4 warehouses
Barnes	32K particles (16 threads)
Lu	2048×2048 matrix (16 threads)
Bodytrack	4 frames and 1K particles (16 threads)
Fluidanimate	5 frames and 300K particles (16 threads)
Swaptions	64 swaptions and 20K simulations (16 threads)
MIX1	Hmmer (reference) (16 copies)
MIX2	Barnes, Ocean (514×514 grid), Radix (3M int), Lu, Milc (ref), Mcf (ref), Bzip2 (ref), and Hmmer (2 threads/copies each)
MIX3	Barnes, FFT (4M complex numbers), Lu, and Radix (3M int) (4 threads each)

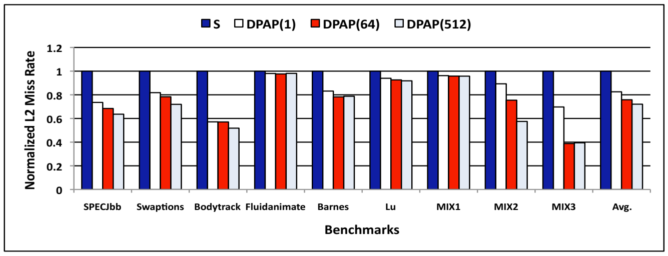

Fig. 3. **L2 miss rates of the baseline shared scheme (S) and DPAP (normalized to S).**

We use a mixture of multithreaded and multiprogramming workloads from SPLASH-2 [20], SPEC2006 [18], and PARSEC [1] suites. Table 2 shows the data sets and other important features of the simulated workloads. Lastly, the programs are fast forwarded to get past of their initialization phases. After various warm-up periods, each SPLASH-2 and PARSEC benchmark is run until the completion of its main loop, and each of SpecJBB, MIX1, MIX2, and MIX3 is run for 8 billion user instructions.

4.2 Results

Let us first compare DPAP against the baseline shared scheme (S). DPAP offers a systematic solution to reduce interference misses in distributed shared caches. Fig. 3 shows the L2 miss rates of S, DPAP(1), DPAP(64), and DPAP(512) normalized to S. DPAP(1), DPAP(64), and DPAP(512) correspond to DPAP running with 1-group, 64-group, and 512-group granularities. We actually ran DPAP with all possible group granularities but show only three configurations. Dividing a bank into only 64 groups (i.e., a counter per each group where each group encompasses 8 sets) provides close benefits as compared to having 512 groups (i.e., a counter per each set). On average, DPAP(1), DPAP(64), and DPAP(512) achieve L2 miss rate reductions of 17.4%, 24.1%, and 27.9% over S, respectively.

We can improve cache performance not only by efficient cache management but also via increasing cache size and associativity. To compare against these varieties, we augment each cache set of S with two more ways (i.e., 64KB of capacity) and double the L2 cache size (i.e., 1MB instead of 512KB per each L2 bank). We refer to the first

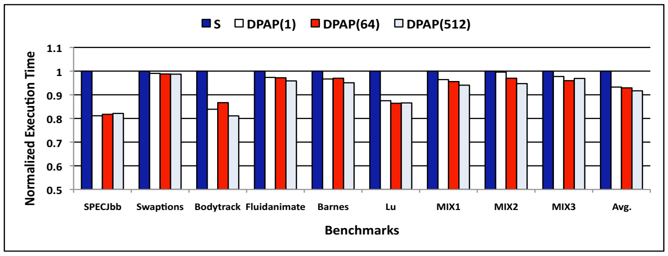

Fig. 4. **Execution times of the baseline shared scheme (S), DPAP(1), DPAP(64), and DPAP(512) (normalized to S).**

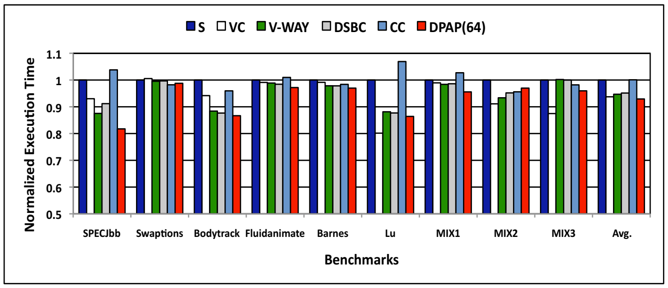

Fig. 5. **Execution times of the baseline shared scheme (S), victim cache (VC), variable-way set associative cache (V-WAY), dynamic set balancing cache (DSBC), cooperative caching (CC), and DPAP(64) (all normalized to S).**

and latter configurations as S(2W) and S(D), respectively. We ran S(2W) and S(D) and found that they accomplish average L2 miss rate reductions of 4.3% and 11.9% over S, respectively. We conclude that DPAP is quite attractive since with small design and storage overhead, it provides average miss rate reduction benefits over S with twice its cache size (i.e., S(D)).

To that end, Fig. 4 presents the execution times of S, DPAP(1), DPAP(64), and DPAP(512) normalized to S. DPAP(1), DPAP(64), and DPAP(512) outperform S by averages of 6.7%, 7%, and 8.3%, respectively. DPAP can potentially place blocks closer (or further) to requester cores than S and, accordingly, reduce (or increase) the average L2 access latency (AAL). For instance, although DPAP(512) achieves more miss rate reduction than DPAP(64) for SpecJBB, DPAP(64) outperforms DPAP(512). As another example, Lu exhibits a performance improvement of 13.5% with only 7.4% L2 miss rate reduction under DPAP(64). We found that DPAP(64) achieves a 8.8% AAL improvement over S for Lu.

Finally, we compare DPAP against victim caching (VC) [6], V-WAY [11], DSBC [13], and CC [2]. Fig. 5 depicts the execution times of all the compared schemes normalized to S. We consider only DPAP(64) and set the cooperation throttling probability of CC to 70%. Besides, we consider a fully associative 16KB VC per tile. We *optimistically* assume only a 6 cycle access time to VC after each miss on an L2 bank. VC extends the associativity of hot sets in the cache. Nonetheless, when multiple host sets compete for a VC space, VC is flushed quickly and fails subsequently to reduce capacity and interference misses appreciably (e.g., Swaptions). For the simulated benchmarks, DPAP(64) outperforms VC by an average of 0.5%.

The V-WAY cache centers around the use of global replacement for line refills. However, the global replacement policy of V-WAY is triggered only if an invalid tag entry is found upon a miss. Otherwise, V-WAY bypasses the global policy and identifies a tag victim using a local LRU policy. Besides, DSBC offers a very limited sharing across local cache sets and allows only a unidirectional retention between source and destination sets. V-WAY and DSBC reduce the L2 miss rate of S by averages of 14.7% and 11.3%, respectively. Lastly, CC spills cache blocks to neighboring L2 banks without knowing if spilling helps or hurts the miss rate [10]. Hence, CC might sometimes degrade the system performance (e.g., SpecJBB). CC is based on the private CMP scheme. CC improves the L2 miss rate of P by an average of 2.1% but degrades AAL by 2.5%. On average, DPAP(64) outperforms V-WAY, DSBC, and CC by 1.8%, 2.3%, and 6.9%, respectively.

5 Conclusion and Future Work

Crossing the billion-transistor per chip barrier has had a profound influence on the emergence of chip multiprocessors (CMPs) as a mainstream architecture of choice. This paper investigates the interference misses problem in distributed shared CMP caches and proposes dynamic pressure-aware associative placement (DPAP), a novel strategy that reduces the non-uniform distribution of memory accesses across aggregate cache sets. Temporal pressure information (how many lines yield cache hits during a time interval) is collected at a group (composing of local cache sets) granularity and recorded in an array at the memory controller. On an incoming cache block, DPAP inspects the pressure array, identifies the tile with the minimum pressure, and places the block at that tile.

We set forth two main future directions. First, we will incorporate more parameters to the placement process. Specifically, we will target, in addition to the interference misses problem, the NUCA latency problem. Finally, we will employ more accurate pressure measurements. For instance, *spatial* pressure (how many *unique* lines yield cache hits during a time interval), rather than only temporal, can be involved with DPAP.

References

[1] C. M. Bienia, S. Kumar, J. P. Singh, and K. Li. “The PARSEC Benchmark Suite: Characterization and Architectural Implications,” *PACT*, 2008.

[2] J. Chang and G. S. Sohi. “Cooperative Caching for Chip Multiprocessors,” *ISCA*, 2006.

[3] M. H. Hammoud, S. Cho, and R. Melhem. “ACM: An Efficient Approach for Managing Shared Caches in Chip Multiprocessors ,” *HiPEAC*, 2009.

[4] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. “Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches,” *ISCA*, 2009.

[5] HP Labs. “http://www.hpl.hp.com/research/cacti/”

[6] N. P. Jouppi. “ Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers,” *ISCA*, 1990.

[7] C. Kim, D. Burger, and S. W. Keckler. “An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches,” *ASPLOS*, 2002.

[8] P. Kongetira, K. Aingaran, and K. Olukotun. “Niagara: A 32-Way Multithreaded Sparc Processor,” *IEEE MICRO*, 2005.

[9] K. Olukotun, L. Hammond, and J. Laudon. “Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency ,” *Synthesis Lectures on Computer Arch*, 1st Ed., Morgan and Claypool, 2007.

[10] M. K. Qureshi. “Adaptive Spill-Receive for Robust High-Performance Caching in CMPs,” *HPCA*, 2009.

[11] M. K. Qureshi, D. Thompson, and Y. N. Patt. “The V-WAY Cache: Demand-Based Associativity via Global Replacement,” *ISCA*, 2005.

[12] Research at Intel. “Introducing the 45nm Next-Generation Intel Core^{TM} Microarchitecture,” *White Paper.*

[13] D. Rolán, B. B. Fraguela, and R. Doallo “Adaptive Line Placement With the Set Balancing Cache,” *MICRO*, 2009.

[14] A. Ros, M. E. Acacio, and J. M. García “Scalable Directory Organization for Tiled CMP Architectures,” *CDES*, 2008.

[15] T. Sherwood, B. Calder, and J. Emer. “Reducing CacheMisses Using Hardware and Software Page Placement,” *ICS*, 1999.

[16] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner. “POWER5 System Microarchitecture,” *IBM J. Res. & Dev.*, 49(1):–25, 2005.

[17] S. Srikantaiah, M. Kandemir, and M. J. Irwin. “Adaptive Set Pinning: Managing Shared Caches in Chip Multiprocessors,” *ASPLOS*, 2008.

[18] Standard Performance Eval. Corp. `http://www.specbench.org`.

[19] Virtutech AB. Simics Full System Simulator “http://www.simics.com/”

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. “The SPLASH-2 Programs: Characterization and Methodological Considerations,” *ISCA*, 1995.

[21] M. Zhang and K. Asanović. “Victim Migration: Dynamically Adapting Between Private and Shared CMP Caches,” *TR-2005-064, MIT*, 2005.