C-AMTE: A Location Mechanism for Flexible Cache Management in Chip Multiprocessors

Mohammad Hammoud, Sangyeun Cho, and Rami Melhem

Department of Computer Science
University of Pittsburgh

Abstract

This paper describes Constrained Associative-Mapping-of-Tracking-Entries (C-AMTE), a scalable mechanism to facilitate flexible and efficient distributed cache management in large-scale chip multiprocessors (CMPs). C-AMTE enables fast locating of cache blocks in CMP cache schemes that employ one-to-one or one-to-many associative mappings. C-AMTE stores in per-core data structures tracking entries to avoid on-chip interconnect traffic outburst or long distance directory lookups. Simulation results using a full system simulator demonstrate that C-AMTE achieves improvement in cache access latency by up to 34.4%, close to that of a perfect location strategy.

Key words:
CMP, Shared Scheme, Private Scheme, Associative Mapping, Fixed Mapping, Tracking Entries.

1. Introduction

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. As CMPs' realm is continuously expanding, they must provide high and scalable performance. One of the key challenges to obtaining high performance from CMPs is the management of the limited on-chip cache resources (typically the L2 cache) shared by multiple executing threads.

Tiled CMP architectures have recently been advocated as a scalable design. They replicate identical building blocks (tiles) connected over a switched network on-chip (NoC). A tile typically incorporates a private L1

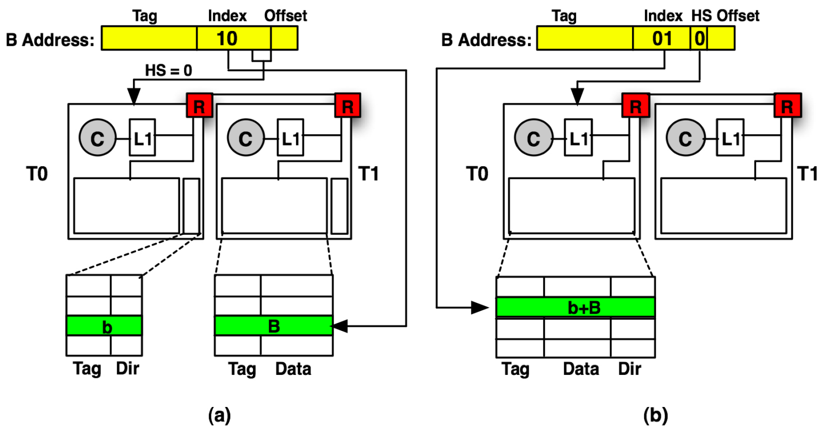


Figure 1: Two traditional cache organizations. Each tile has a direct-mapped L2 cache bank. Each bank incorporates 4 cache blocks. (a) Private scheme. (b) Shared scheme.

cache and an L2 cache bank. A traditional practice, referred to as the private scheme, implies that each processor has its own private L2 cache bank. The effective cache associativity of the private organization equates the aggregate associativity of the L2 cache banks [3]. For instance, 16 private L2 banks with 8-way associativity effectively offers 128-way set associativity. Each tile attracts cache blocks to its local L2 bank. Thus, a cache block can map to any of the 128-way entries, and if shared amongst cores, can reside in multiple L2 banks.

Fig. 1(a) shows a simplified dual core tiled CMP version that demonstrates how a private scheme works. Tile, T1, maps a cache block, B, to its L2 bank. Three options can be employed to maintain coherence among the private L1 and L2 caches: a broadcast-based, a centralized directory, and a distributed directory protocols. The broadcast-based and the centralized protocols are deemed non-scalable especially with the advent of medium-to-large scale CMPs and the projected industrial plans. A high-bandwidth distributed on-chip directory can be adopted to accomplish the task [12]. As depicted in Fig. 1(a), specific bits from the physical address of B, denoted as the home select (HS) bits, are utilized to map a *tracking entry*, b, to a tile, referred to as the *home tile* of B. The tracking entry b consists of B's tag bits and its corresponding directory information. We identify a mapping process that maps an entry (block or tracking entry) to a fixed tile as a *fixed mapping* strategy. On the other hand, we refer to a mapping process that exploits the aggregate associativity of the L2 cache banks as an *associative mapping* strategy. We particularly denote the private scheme process for mapping cache blocks to L2 banks as a *one-to-many associative mapping* policy because a single block might be mapped to multiple banks.

Another traditional practice of CMP caches is a one that logically shares the physically distributed L2 banks. A shared cache design stores a single copy of the shared data at the L2 level. However, shared caches suffer from a growing on-chip delay problem. Access latencies to L2 banks differ essentially depending on the distances between requester cores and target banks. This design is referred to as a *Non Uniform Cache Architecture* (NUCA). Fig. 1(b) illustrates how the shared scheme works. Block B and its corresponding tracking entry b are mapped together to B's home tile, T0, designated by the HS bits of B's physical address. As such, the shared scheme utilizes fixed mapping for both, cache blocks and tracking entries.

To mitigate the NUCA effects, many proposals have extended the nominal basic shared design to allow associative mapping. For instance, block migration [2, 6, 8, 9, 15] exploits associative mapping by moving frequently accessed blocks closer to requesting cores. We expressly refer to the mapping process employed by migration schemes as *one-to-one associative mapping* strategy because they maintain the exclusiveness of cache blocks at the L2 level. Furthermore, block replication [5, 3, 16] utilizes one-to-many associative mapping to alleviate non-uniform access latencies.

A major shortcoming of using associative mapping for blocks in any CMP cache management scheme is the location process. For example, a migration scheme that promotes a cache block B to a tile different than its home tile, denoted as the *current host* of B, can't use anymore the HS bits of B's physical address to locate B. Consequently, different strategies for the location process need to be considered. A tracking entry can always be retained at a centralized directory or at B's home tile (if the underlying directory protocol is distributed) to be able to track B subsequently. Hence, if a core requests B, the repository of the tracking entries is reached first then the query is forwarded to B's host tile to satisfy the request. The disadvantage of this option is the arousal of 3-way cache-to-cache transfers to satisfy requests. This might degrade the average L2 access latency. An alternative location strategy could be to broadcast queries to all the tiles assuming no tracking entry for B is kept at a specific repository. Such a strategy can, however, burden the NoC and potentially degrade the average L2 access latency.

This paper proposes Constrained Associative-Mapping-of-Tracking-Entries (C-AMTE), a mechanism that flexibly accelerates cache management in

CMPs. In particular, C-AMTE presents *constrained associative mapping* that combines the effectiveness of both, the associative and fixed mapping strategies and applies that to tracking entries to resolve the challenge of locating cache blocks without broadcasting and with minimal 3-way communications.

To summarize, the contributions of C-AMTE are as follows:

- It enables fast locating of cache blocks without swamping the NoC.
- It can be applied whenever associative mapping is used for cache blocks, either in case of one-to-one (i.e, migration) or one-to-many (i.e, replication).
- It can be generally applied to cache organizations that extend the traditional private or shared schemes. Furthermore, it opens opportunities for architects to propose more creative cache management designs with no necessity to stick to either private or shared designs.

The rest of the paper is organized as follows. Section 2 presents some recent CMP cache management schemes. C-AMTE mechanism is detailed in Section 3. In Section 4 we evaluate C-AMTE, and we conclude in Section 5.

2. Related Work

As CMP has become the mainstream architecture of choice, many proposals in the literature advocated managing CMP last level caches (typically L2 caches) to obtaining high system performance. Recent works have recognized the need for shared L2 caches that follow NUCA [6, 8, 9]. NUCA design suffers from a latency problem. A cache block may map to a bank far away from the requester core thus causing the core significant latency to locate the block. Data migration and replication have been suggested as techniques to diminish the NUCA negative effects. Replication schemes complicate the coherence protocol and increase L2 miss rate. Contrarily, migration schemes maintain simple coherence protocol and effective L2 miss rate. However, what hinders the migration schemes from being truly effective in tackling the NUCA problem is the location process.

Beckmann and Wood [2] and Huh et al. [8] studied generational promotion and suggested *Dynamic NUCA* (DNUCA) that migrates blocks towards banks close to requesting processors. To locate migratory blocks, [8] adopts sending concurrent queries to L2 banks. To reduce the number of queries

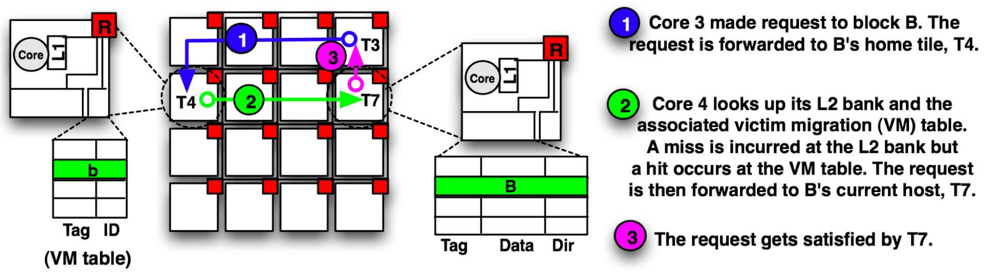

Core
L1
R
T3
T4
T7
1
2
3
b
Tag ID
(VM table)
B
Tag Data Dir
1 Core 3 made request to block B. The request is forwarded to B's home tile, T4.
2 Core 4 looks up its L2 bank and the associated victim migration (VM) table. A miss is incurred at the L2 bank but a hit occurs at the VM table. The request is then forwarded to B's current host, T7.
3 The request gets satisfied by T7.

Figure 2: Locating a migratory block using 3-Way cache-to-cache transfer.

sent over the NoC, [2] staggers the location process by searching each L2 bank sequentially in an increasing order of their distances from the requester core.

Guz et al. [6] presented a new architecture that utilizes migration to divert only *shared data* to cache banks at the center of the chip close to all the cores. To locate migratory blocks, sequential, hybrid (between sequential and broadcast), and sequential with predictor policies have been scrutinized. Kandemir et al. [9] proposed a mechanism that determines a suitable location for a data block, B, within the shared L2 space at any given point during execution. B is then migrated to that suitable place. To locate B, a multistep checking scheme was employed.

Zhang and Asanović [15] examined straightforward promotion and proposed *Victim Migration* that migrates a cache block, B, from its home tile to the initial requester tile. A victim migration (VM) table per tile was suggested to keep track of the locations of migratory blocks. Specifically, a migration tag for B is kept in the VM table at B's home tile to point to the current host of B. Later if a core S reaches the home tile of B and fails to find a matching tag in the regular L2 tag array but hits in the associated VM table, the current host of B, pointed out by the matched migration tag, satisfies the request using 3-way cache-to-cache transfer. Fig. 2 illustrates how a requester core, T3, can locate a cache block, B, that has already been promoted from T4 to T7 manifesting 3-way communication between T3, T4, and T7. A tag, b, has been stored at the home tile of B, T4, to point to the current host of B, T7. Clearly, such a location strategy fails to exploit *distance locality*. That is, the request incurs significant latency to locate B, though B resides in close proximity.

Marty and Hill [10] proposed imposing a two-level virtual coherence hierarchy on a physically flat CMP that harmonizes with virtual machines (VMs) assignments. A key challenge for an intra-VM protocol is finding the home tile of a requested block. For an intra-VM, the home tile is a function of two properties: which tiles belong to a VM and how many tiles belong to a VM. Moreover, dynamic VM reassignment can change both. They suggest co-locating caches with tables within tiles. A table must be looked up before a miss leaves a tile. Each table includes 64 six-bit entries indexed by the six least-significant bits of the block number. Tables would be set by a hypervisor (or OS) at VM (or process) reassignment.

Hammoud et al. [7] proposed an adaptive controlled migration (ACM) scheme that relies on prediction to collect accessibility information regarding cores that accessed a block B in the past, and then assuming that each of these cores will access B again in the future, dynamically migrates B to a bank that minimizes the overall network hops needed. To locate cache blocks, they suggested cache-the-cache-tag (CTCT) location policy. CTCT is a specific version of the C-AMTE mechanism and had been presented in [7] particularly to perform blocks' locations for ACM. This paper presents C-AMTE, a general version of CTCT and can be utilized by any CMP cache management scheme that adopts one-to-one (i.e., migration) or one-to-many (i.e., replication) associative mapping.

Lastly, many researchers explored data replication instead of migration to mitigate the NUCA latency problem. Zhang and Asanović [16] proposed victim replication (VR) scheme based on the nominal shared design. VR keeps replicas of local primary cache victims within *only* the local L2 cache banks. As such, the location process renders straightforward and can be performed simply by looking up local L2 banks seeking for replica hits to avoid approaching the requested blocks' home tiles. However, many cache schemes don't limit themselves to replicating blocks only at two locations, home tiles and local L2 banks. Much work allows one-to-many associative mapping. To mention some, Chang and Sohi [3] proposed *cooperative caching* based on the private scheme, and created a globally managed shared aggregate on-chip cache. Chisti et al. [5] proposed *CMP-NuRAPID* that controls replication based on usage patterns. [3] and [5] utilize 3-way cache-to-cache transfers to satisfy L2 requests upon misses at local L2 banks.

C-AMTE can be essentially used by any of the above cache schemes to resolve the challenge of locating cache blocks. As [7] presents ACM augmented with the cache-the-cache-tag (CTCT) location policy, we in this paper present a generalized and detailed version of CTCT (C-AMTE), select

	Block Mapping	Tracking Entries Mapping
Private Scheme (P)	Associative	Fixed
Shared Scheme (S)	Fixed	Fixed
Scheme With C-AMTE	Associative	Constrained=Fixed + Associative

Table 1: Varieties mapping strategies for alternative CMP caches.

an alternative scheme to apply C-AMTE to (the DNUCA [2, 8] scheme- multiple others can be selected), and compare additionally with various location options.

3. The Proposed Mechanism

Constrained Associative-Mapping-of-Tracking-Entries (C-AMTE) mechanism offers a robust and versatile location strategy for CMP cache schemes. Assuming a distributed directory protocol is adopted, C-AMTE supports storing at least one tracking entry corresponding to a block B at the home tile of B. We refer to this tracking entry as the *principal* tracking entry. The principal tracking entry points to B and can always be checked by any requester core to locate B at its current host. The principal tracking entry is stored using a fixed mapping strategy because the home tile of B is designated by the HS bits of B's physical address. C-AMTE also supports storing another type of tracking entries for B at requester tiles. We refer to this type of tracking entries as *replicated* tracking entries. A replicated tracking entry at a requester tile also points to the current host of B but can be rapidly checked by a requester core to directly locate B (vis-a-́vis checking with B's home tile to locate B at its current host). The idea of replicating tracking entries at requester tiles capitalizes on the one-to-many associative mapping strategy traditionally applied for cache blocks. C-AMTE combines associative and fixed mapping strategies and apply that to tracking entries in order to efficiently solve the location problem. Table 1 illustrates the hybrid approach of the C-AMTE mechanism. We refer to such a hybrid mapping process as a *constrained associative mapping* strategy.

Based on the above discussion, per each tile, T, a principal tracking entry is kept for each cache block B whose home tile is T but had been mapped to another tile. Besides, replicated tracking entries are retained at T to track the locations of other corresponding cache blocks that have been recently accessed by T but whose home tile is not T. Though both, principal and tracking entries essentially act as pointers to the current hosts of cache blocks, we differentiate between them for consistency and replacement purposes (more on this shortly). Per-core data structures can be added to store the two types of the tracking entries. A data structure, referred to as the principal tracking entries (PTR) table, can hold principal tracking entries, and a data structure, referred to as the replicated tracking entries (RTR) table, can hold replicated ones. Alternatively, a single table, could be referred to as the tracking entries (TR) table, can be added to hold both classes of tracking entries pertaining that a hardware extension (i.e. an indicative bit) is engaged to distinguish between the two entries. Pros and cons for splitting the TR table into RTR and PTR are discussed afterward.

Assume a CMP organization with PTR and RTR tables. Whenever a core issues a request to a block B, its RTR table is checked first for a matching replicated tracking entry. On a miss, the home tile of B is reached and its PTR table is looked up. If a miss occurs at the PTR table, B is fetched from the main memory and placed at a tile specified by the underlying cache scheme protocol. If B is mapped to a tile different than its home tile, a principal tracking entry is stored at the PTR table and a replicated tracking one is retained at the requester's RTR table. If, in contrary, B is mapped to its home tile, no tracking entries are kept at either PTR or RTR tables on the home and the requester tiles, respectively. This is because after an incurred miss at the requester's RTR table, B's home tile can be reached with no need to a replicated tracking entry (by using B's HS bits). Besides, B can be located at its home tile without requiring a principal tracking entry (by concurrently looking up the PTR table and the regular L2 bank). This results in space reduction. On the other hand, if a hit occurs at the PTR table, B is located at its current host tile and a replicated tracking entry is furthermore stored at the requester's RTR table. Lastly, on a hit at the requester's RTR table, B is located directly at its current host designated by the matched replicated tracking entry. This consequently avoids a 3-way cache-to-cache transfer that would have been incurred if we had to approach B's home tile to locate B. Same logic applies if a CMP organization assumes a single TR table instead of split PTR and RTR ones.

Fig. 3 demonstrates an example of the C-AMTE mechanism on a tiled CMP with an assumed shared scheme and a certain migration policy being incorporated to alleviate the NUCA latency problem. Fig. 3(a) shows a request made by core 3 to cache block B. Core 3 looks up it local RTR table. We assume a miss occurs and the request is subsequently forwarded to B's home tile, T12. The PTR table and the regular L2 bank at T12 are looked up concurrently. We assume misses occur. As such, B is fetched from the

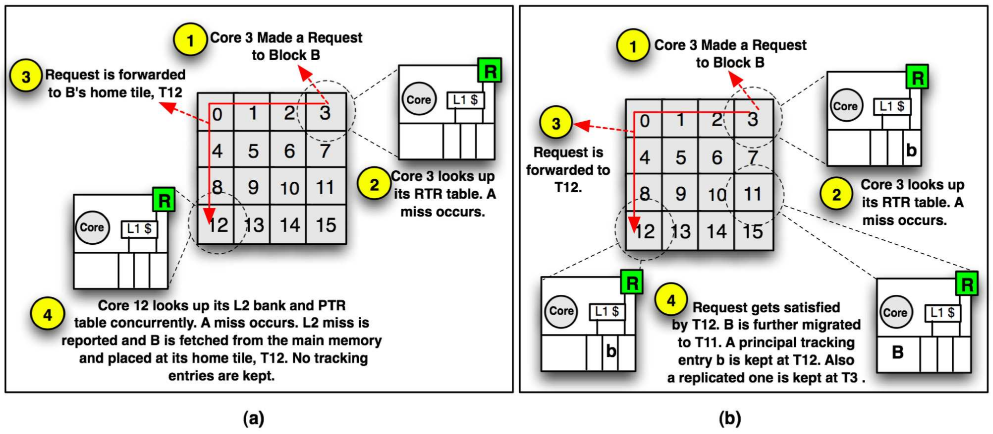
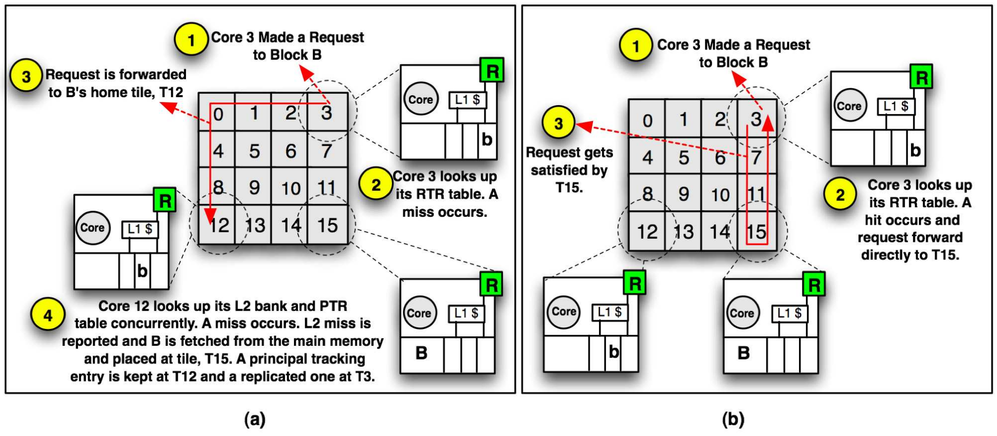

1 Core 3 Made a Request to Block B
3 Request is forwarded to B's home tile, T12
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
Core
L1 $
R
2 Core 3 looks up its RTR table. A miss occurs.
4 Core 12 looks up its L2 bank and PTR table concurrently. A miss occurs. L2 miss is reported and B is fetched from the main memory and placed at its home tile, T12. No tracking entries are kept.
(a)
1 Core 3 Made a Request to Block B
3 Request is forwarded to T12.
b
2 Core 3 looks up its RTR table. A miss occurs.
4 Request gets satisfied by T12. B is further migrated to T11. A principal tracking entry b is kept at T12. Also a replicated one is kept at T3 .
B
(b)

Figure 3: A first example on locating a migratory block B using the C-AMTE mechanism.

main memory and assumedly placed at B's home tile, T12. As discussed previously, we don't store tracking entries for B when B maps to its home tile. B can be directly located at T12 upon any subsequent request (similar to the nominal shared). Fig. 3(b) shows a subsequent request made by core 3 to B. B is located at its home tile, T12. Assume after that hit, B is migrated to T11. Thus, corresponding principal and replicated tracking entries are stored at T12 and T3, respectively. If at any subsequent time core 3 requests B again, a hit will occur at its RTR table (in case the entry has not been replaced yet) and B can be located straightforwardly at T11 avoiding thereby a 3-way cache-to-cache transfer. Lastly, note that if any other core requests B, T12 can be always reached to locate B.

Fig. 4 demonstrates C-AMTE in operation for an assumed scheme that doesn't stick to the traditional placement strategy of the shared design (possibly to mitigate cache conflicts). This is to show the flexibility that C-AMTE can grant to architects when managing CMP caches. Fig. 4(a) shows a request made by core 3 to a cache block B. We assume misses occur at RTR on T3, and PTR and L2 bank on T12, the home tile of B. B is fetched from the main memory and assumedly mapped to T15. Principal and replicated tracking entries are thus kept at T12 and T3, respectively. Fig. 4(b) shows a request made again by core 3 to B. A hit occurs at T3's RTR table. Consequently, B is directly located at T15. Clearly, this exhibits how C-AMTE can allow creativity in blocks placements, avoid cache-to-cache transfers, and exploit distance locality.



1 Core 3 Made a Request to Block B
3 Request is forwarded to B's home tile, T12
2 Core 3 looks up its RTR table. A miss occurs.
4 Core 12 looks up its L2 bank and PTR table concurrently. A miss occurs. L2 miss is reported and B is fetched from the main memory and placed at tile, T15. A principal tracking entry is kept at T12 and a replicated one at T3.
(a)
1 Core 3 Made a Request to Block B
3 Request gets satisfied by T15.
2 Core 3 looks up its RTR table. A hit occurs and request forward directly to T15.
(b)
Core
L1 $
R
b
B

Figure 4: A second example on locating a block B using the C-AMTE mechanism.

The principal and replicated tracking entries need to be kept consistent. We accomplish this by embedding a bit vector with each principal tracking entry at the PTR tables to indicate which tiles stored related replicated tracking entries. For instance, given the example depicted in Fig. 3, each time B is migrated to a different tile, the principal and the replicated tracking entries that correspond to B are updated to point to the new host of B. To that end, each principal tracking entry would include: (1) The tag of the related block (typically 22 bits), (2) a bit vector that acts as a directory to maintain consistency of the principal and the replicated tracking entries (16 bits for a 16-tile CMP model), and (3) an ID that points to the tile that is currently hosting the cache block (4 bits for a 16-tile CMP model). On the other hand, a replicated tracking entry would include only the tag of the related block and the ID of the current host tile. Contrarily, in case of a single TR table, both, the principal and the replicated tracking entries would each comprise of a tag, a bit vector, and an ID fields beside a bit to distinguishe between the two types of entries (required for replacement purposes as discussed afterward). In this case, the bit vector agumented to each replicated entry renders redundant and causes increase in space. As such, splitting TR table to RTR and PTR might be preferable.

Finally, PTR and RTR tables can employ the LRU replacement policy. However, in case of a single TR table, it is wise to never evict a principal tracking entry in favor of a replicated one. An eviction of a principal tracking entry calls for eviction to the corresponding cache block and all the related

Component	Parameter
Cache Line Size	64 B
L1 I/D-Cache Size/Associativity	16KB each/2way
L1 Read Penalty (on hit per tile)	1 cycle
L1 Replacement Policy	LRU
L2 Cache Size/Associativity	512KB per L2 bank or 8MB aggregate/16way
L2 Bank Access Penalty	12 cycles
L2 Replacement Policy	LRU
Latency Per NoC Hop	3 cycles
Memory Latency	300 cycles

Table 2: System parameters

Name	Input
SPECjbb	Java HotSpot (TM) server VM v 1.5, 4 warehouses
Ocean	514×514 grid (16 threads)
Barnes	32K particles (16 threads)
Lu	2048×2048 matrix (16 threads)
Radix	3M integers (16 threads)
MIX1	reference (16 copies of the Spec CPU2006 Hmmer)
MIX2	reference (16 copies of the Spec CPU2006 Sphinx)
MIX3	reference (Barnes, 2 Mcf, 2 Bzip2, 2 Milc, and 2 Hmmer)

Table 3: Benchmark programs

replicated tracking entries. Therefore, the TR replacement policy should replace the following three classes of entries in ascending order: (1) an invalid entry, (2) an LRU replicated tracking entry, (3) and an LRU principal tracking entry. Besides, upon storing a replicated tracking entry, only the first two classes are considered. If no entry belonging to one of these two classes is found, a replicated tracking entry is not retained.

4. Quantitative Evaluation

To demonstrate the potential performance gain of the C-AMTE mechanism, we evaluate the DNUCA [2, 8] scheme with 4 different location strategies and compare against the baseline shared cache design (S). Ideal is a scheme that assumes that cores have oracle knowledge about the cache blocks residences. Broadcast (B) supports sending queries to all the tiles upon upon every L2 request. Three-way (3W) advocates a DNUCA implementation with a distributed directory protocol that employs 3-way cache-to-cache transfers.

Finally, C-AMTE is an implementation of the DNUCA scheme with the C-AMTE mechanism being fully engaged.

4.1. Methodology

We present our results based on a detailed full system simulation using Simics 3.0.29 [1] running under Solaris 10 OS. We fully developed our own CMP cache module including a 2D mesh NoC model. We simulate a tiled CMP architecture comprised of 16 UltraSPARC-III Cu processors. Each processor uses an in-order core model. The tiles are organized as 4×4 grid connected by the 2D mesh NoC. Each tile encompasses a switch, an aggregate 32KB I/D L1 cache, a 512KB L2 cache bank, and a TR table with 16K entries. The latency to lookup a TR table is hidden under the delay to enqueue the request in the port scheduler of the local switch [4]. The dimension-ordered (XY) routing algorithm is employed where messages are first routed in the X and then the Y directions. All types of messages are modeled to consume NoC bandwidth. For coherence enforcement at the L1 cache level, we model a distributed MESI-based directory protocol. Table 2 shows our configuration's experimental parameters.

To conduct the study we utilize a mixture of multithreaded and multiprogramming workloads. For multithreaded workloads we use the commercial benchmark SPECJBB in addition to four shared memory programs from the SPLASH2 suite [14] (Ocean, Barnes, Lu, and Radix). We also composed three multiprogramming workloads using applications from SPLASH2 and SPEC2006 [13] (Hmmer, Sphinx, Milc, Mcf, and Bzip2). Table 3 shows the data set and other important features of each of the 8 simulated workloads. Lastly, we ran Ocean, Barnes, Lu, and Radix in full and stopped the remaining benchmarks after a detailed simulation of 20 billion instructions.

4.2. Results

Fig. 5 presents the relative reduction/increase in average L2 access latency (AAL) for B, 3W, C-AMTE, and Ideal over S. Latency for an L2 access can be defined in terms of three types of scenarios. First, it can be simply a function of only L2 access time and that would occur upon a hit to a local L2 bank. Second, it can be a function of NoC congestion, Manhattan distance between the requester and the target tiles, and L2 access time. That would occur upon a hit to a remote L2 bank. Third, it can incur a memory access and that would occur upon a miss to L2. C-AMTE achieves AAL improvement over S by an average of 18.6%, and to an extent of 34.4% for

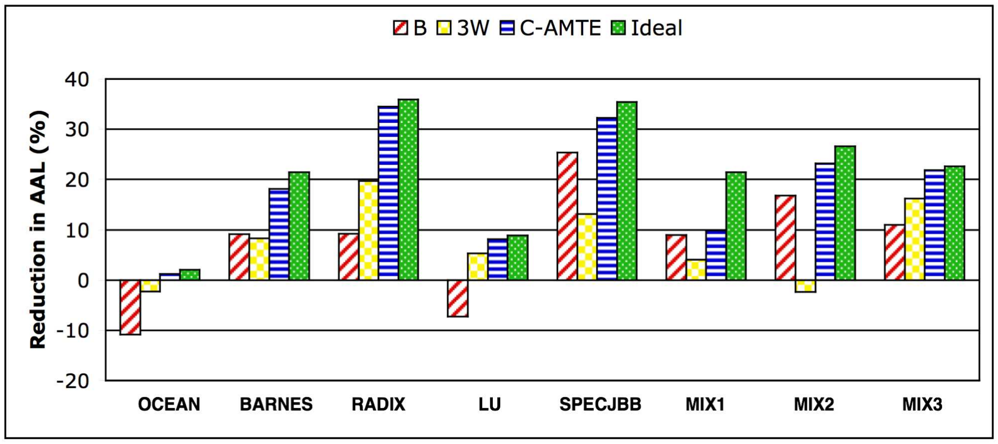


Figure 5: Average L2 Access Latency (Relative reduction over S).

	S	B	3W	C-AMTE	Ideal
Ocean	2.5	35.8	3	3.1	2.4
Barnes	3.6	55.1	4.4	4	2.9
Radix	6.9	136.4	9.8	13.5	9.4
Lu	70	905.4	78.3	76	70.5
SPECjbb	5.3	87.5	5.7	4.8	2.4
MIX1	35.5	573.8	37.3	37.6	27.4
MIX2	22.1	370.2	32.6	47	19
MIX3	16.5	314.6	20.7	19.5	16.9

Table 4: Message-Hops per 1K insructions

Radix. This makes C-AMTE significantly close to Ideal that accomplishes, on the other hand, an average of 21.7% AAL improvement over S. Contrary to that, B reduces AAL by an average of only 7.7% over S, and by as much as -10.8% and 25.3% for Ocean and SPECjbb, respectively. Although B locates cache blocks straightforwardly, it profoundly outbursts the NoC with superfluous queries. It has been found that B increases the on-chip network traffic by $\approx 15x$ over S. Table 4 shows the message-hops per 1K instructions for the simulated applications. An increase in the link congestion between two adjacent switches on the CMP platform escalates the connectivity delay of an L2 request. This deteriorates the consequent AAL. If the connectivity deterioration is not offset by the gain from direct locations to cache blocks, AAL might degrade. This explains the B behavior with the two scientific applications, Ocean and Lu. C-AMTE also increases the on-chip network

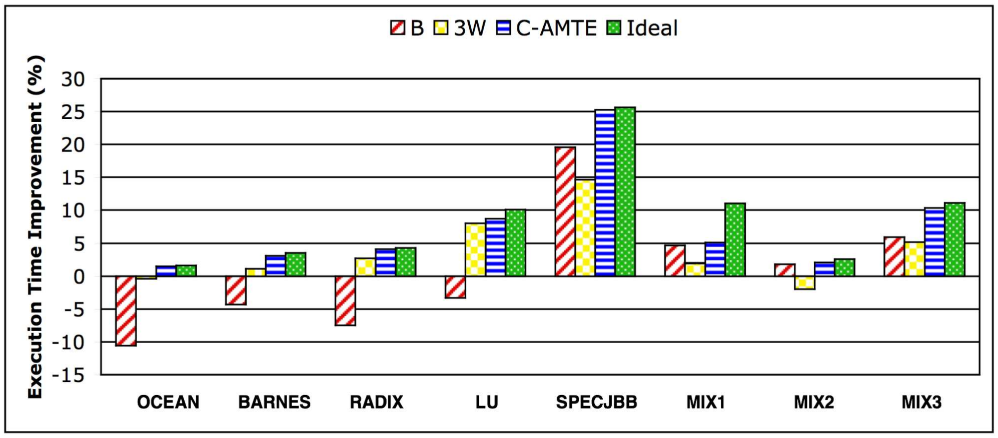


Figure 6: Relative execution time improvement over S.

traffic. The two reasons of that increase are misses to RTR tables by requester cores, and consistency maintenance of the principal and replicated tracking entries upon migrations.

The 3W strategy also offers an average AAL reduction of 7.7% over S. In fact, 3W is not supposed to surpass S because it triggers 3-way cache-to-cache transfers for migratory blocks. Nonetheless, for some benchmarks like Radix, 3W improves the AAL by 19.6% over S though it degrades the on-chip network traffic by 42%. We found that this outshine came from a 32% L2 miss rate decline as compared to S. We observed that Radix has a great deal of L2 misses caused by heavy cores interferences on cache sets (inter-processor misses). In some cases, 3W inadvertently redistributes the cache blocks on L2 and diminishes accordingly the pressure on some of the cache sets.

Finally, Fig. 6 presents the relative speedup/slowdown in total execution time over S. Across all benchmarks, C-AMTE achieves superiority over S by an average of 7.5% (25.2% for the SPECjbb benchmark) making it notably close to Ideal that improves the execution time by an average of 8.7% over S. The speedup of B over S is on average 0.7%. Though B accomplishes 9% and 9.2% AAL reductions over S for Barnes and Radix respectively, this is not effectively realized on the resulted performance. Lastly, the 7.7% average AAL improvement of 3W over S translates to an average speed up of 3.9%.

5. Concluding Remarks

Cache management in CMP is crucial to fuel its performance growth. This paper proposes a mechanism to effectively simplify the process of locating cache blocks of schemes that employ either one-to-one or one-to-many associative mapping. Particularly, C-AMTE stores tracking entries that correspond to cache blocks at per-core data structures for straightforward locations at subsequent accesses. We demonstrated the potential of C-AMTE with an implementation of the DNUCA [2, 8] migration scheme. A performance speedup of up to 25.2% has been achieved, significantly close to a perfect location strategy. Lastly, having established the generality and the effectiveness of C-AMTE, optimizations to reduce hardware complexity is one of the obvious next steps to explore in future.

References

[1] Virtutech AB. Simics Full System Simulator "http://www.simics.com/"

[2] B. M. Beckmann and D. A. Wood. "Managing Wire Delay in Large Chip-Multiprocessor Caches," *MICRO*, Dec. 2004.

[3] J. Chang and G. S. Sohi. "Cooperative Caching for Chip Multiprocessors," *ISCA*, June 2006.

[4] M. Chaudhuri. "PageNUCA: Selected Policies for Page-grain Locality Management in Large Shared Chip-multiprocessor Caches," *HPCA*, pp. 227-238, Feb. 2009.

[5] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. "Optimizing Replication, Communication, and Capacity Allocation in CMPs," *ISCA*, June 2005.

[6] Z. Guz, I. Keidar, A. Kolodny, U. C. Weiser. "Utilizing Shared Data in Chip Multiprocessors. with the Nahalal Architecture," *SPAA*, June 2008.

[7] M. Hammoud, S. Cho, and R. Melhem. "ACM: An Efficient Approach for Managing Shared Caches in Chip Multiprocessors ," *HiPEAC*, pp. 319–330, Jan. 2009.

[8] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. "A NUCA Substrate for Flexible CMP Cache Sharing," *ICS*, June 2005.

[9] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son. "A Novel Migration-Based NUCA Design for Chip Multiprocessors," *SC*, Nov. 2008.

[10] M. R. Marty and M. D. Hill. "Virtual Hierarchies to Support Server Consolidation," *ISCA*, June 2007.

[11] N. Rafique, W. Lim, M. Thottethodi. "Architectural Support for Operating System-Driven CMP Cache Management ," *PACT*, Sep. 2006.

[12] A. Ros, M. E. Acacio, and J. M. García "Scalable Directory Organization for Tiled CMP Architectures," *CDES*, July 2008.

[13] Standard Performance Evaluation Corporation. `http://www.specbench.org`.

[14] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations," *ISCA*, pp. 24–36, July 1995.

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

[16] M. Zhang and K. Asanović. "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," *ISCA*, June 2005.