Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Research note

C-AMTE: A location mechanism for flexible cache management in chip multiprocessors

Mohammad Hammoud *, Sangyeun Cho, Rami Melhem

Department of Computer Science, University of Pittsburgh, United States

A R T I C L E   I N F O

Article history:
Received 17 November 2009
Received in revised form
13 November 2010
Accepted 16 November 2010
Available online 14 December 2010

Keywords:
CMP
Shared scheme
Private scheme
Associative mapping
Fixed mapping
Tracking entries

A B S T R A C T

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.

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/processes.

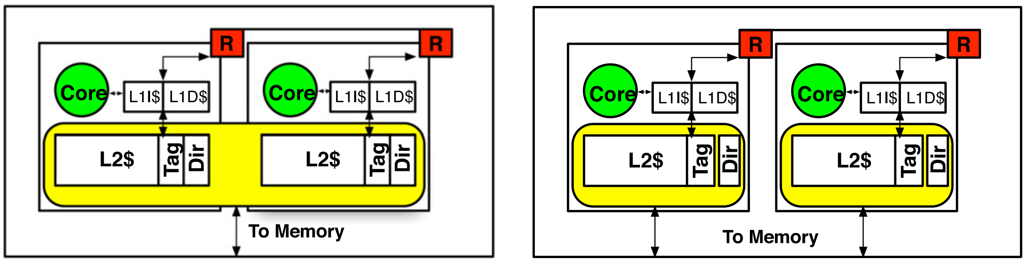
Economic, manufacturing, and physical design considerations suggest tiled CMP architectures (e.g., Tilera's Tile64 and Intel's Teraflops Research Chip) that co-locate distributed cores with distributed cache banks in tiles communicating via a network on-chip (NoC) [12]. A tile typically includes a core, private L1 caches (I/D), and an L2 cache bank. A traditional practice, referred to as the shared scheme, logically shares the physically distributed L2 banks. On-chip access latencies differ depending on the distances between requester cores and target banks creating a Non Uniform Cache Architecture (NUCA) [15]. As an example, the Intel Core^{TM} i7 processor introduces NUCA into its platform [22]. Another conventional practice referred to as the private scheme, associates each L2 bank to a single core and provides no capacity sharing among cores. Fig. 1 demonstrates the two designs. For simplicity we show only a dual-core tiled CMP architecture. In addition, we assume a distributed directory protocol.

The private scheme replicates cache blocks at the L2 banks of the requesting cores. Hence, an effective cache associativity which equates the aggregate associativity of the L2 cache banks is provided [5]. That is, a cache block can map to any of the private L2 banks, and if shared amongst cores, can reside in multiple L2 banks. A high bandwidth on-chip directory protocol can be employed to keep the multiple L2 banks coherent. The directory can be held as a duplicate set of L2 tags distributed across tiles by the set index [1,30]. We generally refer to a mapping process that exploits the aggregate associativity of the L2 cache banks as an *associative mapping* strategy. In particular, we designate the mapping strategy of the private scheme as *one-to-many associative mapping* because a single block can be mapped to multiple L2 banks.

In contrast to the private design, the shared scheme maintains the exclusiveness of cache blocks at the L2 level. A core maps and locates a cache block, B, to and from a target L2 bank at a tile referred to as the *static home tile* (SHT) of B. The SHT of B is determined by a subset of bits denoted as *home select* bits (or HS bits) from B's physical address. As such, the shared strategy requires maintaining coherence only at the L1 level. The SHT of B can store B itself and a bit vector indicating which cores had cached copies of B in their L1 private caches. This on-chip coherence practice is referred to as an in-cache coherence protocol [4,11,30]. In this work we refer to an entry that tracks copies (either at L1 or L2) of a certain cache block as a *tracking entry*. We, furthermore, identify a mapping process that maps an entry (block or tracking)

* Corresponding author.
E-mail address: mhh@cs.pitt.edu (M. Hammoud).

0743-7315/$ – see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2010.11.009



(a) Shared L2 design. (b) Private L2 design.

Fig. 1. Two traditional cache organizations. (a) The shared L2 design backs up all the L1 caches. (b) The private L2 design backs up only the private L1 cache on each tile. (Dir stands for directory and R for router.)

to a fixed tile as a *fixed mapping* strategy (e.g., the shared design employs fixed mapping).

Recent research work on CMP cache management has recognized the importance of the shared scheme [24,11,10,13,26,14]. Besides, many of today's multi-core processors, the Intel Core^{TM}2 Duo processor family [20], Sun Niagara [16], and IBM Power5 [23], have featured shared caches. A shared design, however, suffers from a growing on-chip delay problem. Access latencies to L2 banks are non-uniform and proportional to the distances between requester cores and target banks. This drawback is referred to as the *NUCA problem*.

To mitigate the NUCA problem, many proposals have extended the nominal basic shared design to allow associative mapping (i.e., leveraging the aggregate associativity of the L2 cache banks). For instance, block migration [2,10,13,14,29] exploits associative mapping by moving frequently accessed blocks closer to requesting cores. We denote such a strategy as *one-to-one associative mapping* due to the fact that the exclusiveness of cache blocks at the L2 level is still preserved (only a single copy of a block is promoted along identical sets over different banks). In contrast to migration, replication duplicates cache blocks at different L2 banks [7,5,30]. Accordingly, a replication scheme is said to adopt *one-to-many associative* mapping.

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, cannot use the HS bits of B's physical address to locate B anymore. 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 enable tracking B after promotion. 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 which can 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 overall system performance.

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 location 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 conventional 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 traditional paradigms.

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

2. Related work

Much work has been done to effectively manage CMP caches. Many proposals advocate CMP cache management at either fine (block) or coarse (page) granularities and base their work on either the nominal shared or private schemes. We briefly discuss below some of the prior work and describe the location process that each proposal employs. *We note that C-AMTE is not an independent CMP scheme that can be run by itself, but yet a location mechanism that can be applied to CMP designs that employ one-to-one or one-to-many associative mapping.*

Beckmann and Wood [2] and Huh et al. [13] studied generational promotion and suggested *Dynamic NUCA* (DNUCA) that migrates blocks towards banks close to requesting processors. To locate migratory blocks, [13] adopts sending concurrent queries to L2 banks. To reduce the number of queries sent over the NoC, [2] staggers the location process by searching L2 banks sequentially in an increasing order of their distances from the requester cores.

Guz et al. [10] 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. [14] proposed a mechanism that determines a suitable location for a data block, B, within the shared L2 space at any given point during execution and then migrates B to that suitable place. To locate B, a multistep checking scheme was employed.

Zhang and Asanović [29] examined direct promotion (upon first touch) 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 sharer 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 a 3-way cache-to-cache transfer. Clearly, Victim Migration fails to exploit *distance locality*. That is, the request of a sharer core S might incur significant latency to locate B (due to approaching B's home tile), though B might reside in close proximity to S.

Table 1
Mapping strategies of private and shared CMP caches and the hybrid mapping approach of C-AMTE.

	Block mapping	Tracking entries mapping
Private scheme (P)	Associative (at requesting tiles)	Fixed (at home tiles)
Shared scheme (S)	Fixed (at home tiles)	Fixed (at home tiles)
Scheme with C-AMTE	Associative (one-to-one or one-to-many depending on the underlying cache scheme)	Constrained = Fixed (at home tiles) + Associative (at requesting tiles)

Marty and Hill [18] 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 to find 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. Awkwardly, a dynamic VM reassignment can change both. As such, 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 a VM (or process) reassignment.

Hammoud et al. [11] 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, the cache-the-cache-tag (CTCT) location policy has been suggested. CTCT is a specific version of the C-AMTE mechanism and had been presented in [11] specifically to perform blocks' locations for ACM. This paper generalizes CTCT (now C-AMTE) to enable fast locating of cache blocks in CMP cache schemes that adopt one-to-one (i.e., migration) or one-to-many (i.e., replication) associative mappings.

Cho and Jin [8] proposed an OS-based page allocation algorithm applicable to NUCA architectures. Cache blocks are mapped and located to L2 banks using interleaving on page frame numbers. Chaudhuri [6] suggested PageNUCA which employs data migration at page granularity. Hardvellas et al. [12] presented R-NUCA that relies on the OS to classify cache accesses into either private, shared, or instructions and then places and locates each differently at the L2 cache space. Both, PageNUCA and R-NUCA adopt direct location strategies similar to C-AMTE. In Section 3.5 we detail the two schemes and compare and contrast them versus C-AMTE.

Lastly, many researchers explored data replication instead of migration to mitigate the NUCA latency problem. Zhang and Asanović [30] proposed a 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 becomes straightforward: local L2 banks are looked up (seeking replica hits) before potentially checking with blocks' home tiles. However, many other cache schemes do not limit themselves to replicating blocks at only local L2 banks. Chang and Sohi [5] proposed *cooperative caching* based on the private scheme, and created a globally managed shared aggregate on-chip cache. Chisti et al. [7] proposed *CMP-NuRAPID* that controls replication based on usage patterns. Both, [5,7] utilize 3-way cache-to-cache transfers to satisfy L2 requests upon misses at local L2 banks.

3. The proposed mechanism

3.1. Description of the mechanism

Constrained Associative-Mapping-of-Tracking-Entries (C-AMTE) is not an autonomous CMP cache organization that can run by itself but rather a mechanism that can be applied to CMP cache designs that employ one-to-one (i.e., migration) or one-to-many (i.e., replication) associative mappings. A shared NUCA architecture maps and locates a cache block, B, to and from a *home tile* determined by a subset of bits (home select or HS bits) from B's physical address. Accordingly, B might be mapped to a bank far away from the requester core, causing the core significant latency to locate B. Such a problem is referred to as the *NUCA problem*. Migration and replication have been suggested as techniques to alleviate the NUCA problem. To save latency on subsequent requests to B, migration and replication relocate and replicate, respectively B at a tile different than its home tile, denoted as the *host tile* of B, closer to requesting cores. Consequently, B can have, in addition to the home tile, one or more host tiles. To locate B at a host tile, the HS bits of B's physical address cannot be used anymore. C-AMTE offers a robust and versatile location strategy to locate B at host tiles.

Assuming a distributed directory protocol, C-AMTE supports storing 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 entry for B at requester tiles. We refer to these 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 (instead of checking with B's home tile to achieve that). 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 applies that to tracking entries in order to efficiently solve the location problem. Table 1 illustrates the hybrid approach adopted by the C-AMTE mechanism. We refer to such a hybrid mapping process as a *constrained associative mapping* strategy.

Based on the above discussion, per tile, T, a principal tracking entry is kept for each cache block B whose home tile is T but has been mapped/promoted 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). We can add two distinct data structures per each tile 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.

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. C-AMTE then proceeds as follows:

- On a miss at the RTR table, 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 mapped to a tile T specified by the underlying

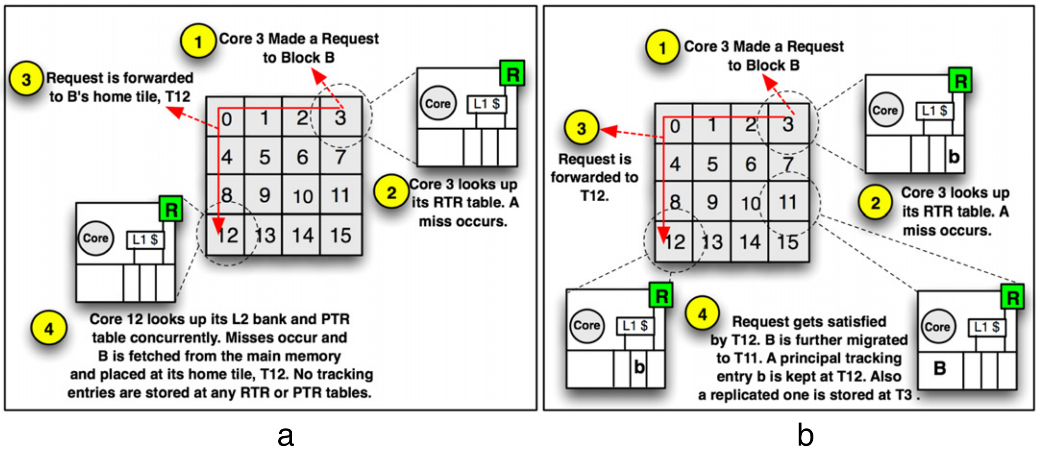

1 Core 3 Made a Request to Block B
3 Request is forwarded to B's home tile, T12
2 Core 3 looks up its RTR table. A miss occurs.
4 Core 12 looks up its L2 bank and PTR table concurrently. Misses occur and B is fetched from the main memory and placed at its home tile, T12. No tracking entries are stored at any RTR or PTR tables.
a
1 Core 3 Made a Request to Block B
3 Request is forwarded to T12.
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 stored at T3 .
b
Core
L1 $
R

Fig. 2. A first example on locating a migratory block B using the C-AMTE mechanism.

cache scheme protocol. If T is not B's home tile, principal and replicated tracking entries are stored at the PTR table of B's home tile and the RTR table of the requester core, respectively. If, on the contrary, T is B's home tile, no tracking entries are kept at either the PTR or RTR tables (B can be located directly using the HS bits of B's physical address).
  - If, on the other hand, a hit occurs at the PTR table, B is located at its current host tile and a replicated tracking entry is stored at the requesters RTR table.
- On a hit at the RTR table, B is located directly at its current host designated by the matched replicated tracking entry.

Therefore, upon a hit to the requesters RTR table, a 3-way cache-to-cache transfer, which would have been incurred if we had to approach B's home tile to locate B, is avoided. A similar logic applies if C-AMTE assumes a single TR table instead of two distinct PTR and RTR ones.

3.2. *Illustrative examples*

Fig. 2 demonstrates an example of the C-AMTE mechanism on a tiled CMP platform, assuming an underlying shared scheme and a migration policy that promotes cache blocks towards requesting cores. Fig. 2(a) shows a request made by core 3 to a cache block, B. Core 3 looks up its 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 at both. Consequently, B is fetched from the main memory and mapped to B's home tile, T12 (following the mapping strategy of the nominal shared scheme). As such, no tracking entries are retained at either PTR or RTR tables. Fig. 2(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 (closer to T3). Thus, corresponding principal and replicated tracking entries are stored at T12 and T3, respectively. If at any later time core 3 requests B again, a hit will occur at its RTR table (as long as the entry has not been replaced yet) and B can be located straightforwardly at T11 avoiding thereby 3-way cache-to-cache transfers. Lastly, note that if any other core requests B, T12 can always indirectly satisfy the request and a corresponding tracking entry can be stored at the new requesters RTR table.

Fig. 3 demonstrates C-AMTE in operation assuming a cache scheme that might map cache blocks to tiles different than their home tiles. Fig. 3(a) shows a request made by core 3 to a cache block B. Core 3 looks up its 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 and misses are then incurred. Consequently, B is fetched from the main memory and mapped to T15 (determined by the mapping strategy of the cache scheme). As such, principal and replicated tracking entries are kept at T12 and T3, respectively. Fig. 3(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, the two examples shown in Figs. 2 and 3 reveal the efficiency and versatility of C-AMTE as a strategy that exploits distance locality. C-AMTE, in fact, opens opportunities for architects to propose creative block migration, replication, and placement CMP strategies with the required location process being on-hand.

3.3. *Maintenance and coherence of the tracking entries*

The principal and replicated tracking entries need to be kept coherent. We accomplish this by embedding a bit vector with each principal tracking entry at the PTR tables to indicate which cores had cached related replicated tracking entries at their RTR tables (much similar to the in-cache coherence protocol in [4]). For instance, given the example depicted in Fig. 2, 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. Besides, C-AMTE can easily preclude potential false misses that can occur when L2 requests fail to hit on cache blocks because they are in transit between L2 banks. When migration is to be performed, a copy, B', of the cache block B is kept at the current bank so as if an L2 request arrives while B is in transit, the request is immediately satisfied without incurring any delay. When B reaches the new host, an acknowledgement message is sent back to the old host to discard B'. The old host keeps track of any tile that accesses B', and when receiving the acknowledgment message, sends an update message to the new host to indicate the new sharers that requested B while it was in transit. The directory state entry of B is consecutively updated. Clearly, enforcing coherence among tracking entries and precluding false misses impose traffic overhead on the network on-chip. Section 4.2 demonstrates the increase in message hops per 1 K instructions incurred by the C-AMTE mechanism.

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 (this is the reason of why we suggested distinguishing between the two entries). An eviction of a principal tracking entry causes evictions to the corresponding cache block and all the related replicated tracking entries. Therefore, the TR replacement policy should replace the following three classes of entries in an ascending order: (1) an invalid entry, (2) an LRU replicated tracking entry, and (3) an LRU principal tracking entry. Besides, upon storing a

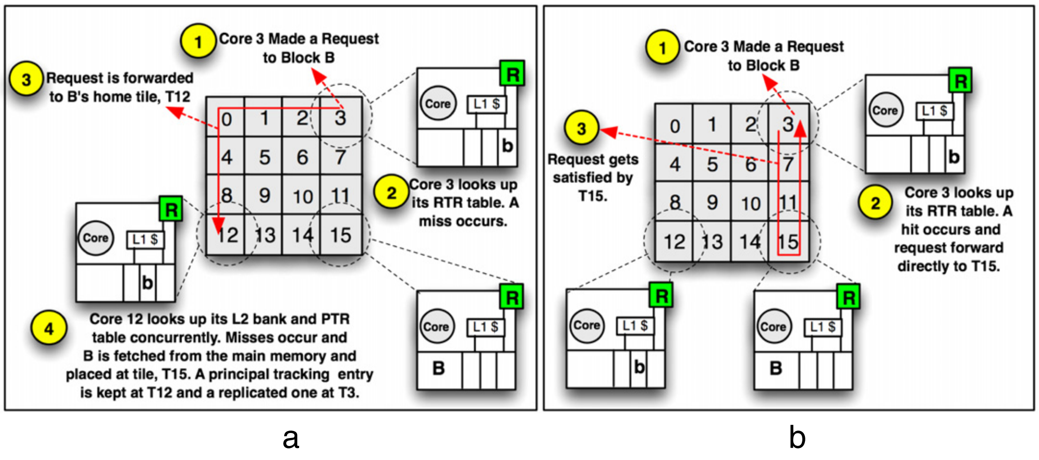

1 Core 3 Made a Request to Block B
3 Request is forwarded to B's home tile, T12
2 Core 3 looks up its RTR table. A miss occurs.
4 Core 12 looks up its L2 bank and PTR table concurrently. Misses occur and B is fetched from the main memory and placed at tile, T15. A principal tracking entry is kept at 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

Fig. 3. A second example on locating a block B using the C-AMTE mechanism.

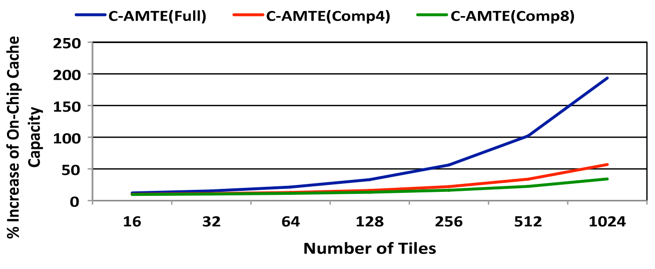


Fig. 4. Storage requirements of C-AMTE with a full-map bit vector (C-AMTE(Full)), a compact vector with 1 bit for every 4 cores (C-AMTE(Comp4)), and a compact vector with 1 bit for every 8 cores (C-AMTE(Comp8)).

replicated tracking entry, only the first two classes are considered for replacement. If no entry belonging to one of these two classes is detected, a replicated tracking entry is not retained.

3.4. Hardware cost and scalability

The storage overhead incurred by C-AMTE pertains to the usage of principal and replicated tracking entries. As described earlier, C-AMTE incurs at least one principal and one replicated tracking entry per cache block, B, when placed at a tile different than its static home tile. On the other hand, C-AMTE incurs at most $N - 1$ tracking entries (one principal and the rest are replicated) per B with N being the number of tiles on the CMP platform. The worst case scenario occurs only when B exhibits a sharing degree of N. Assuming split tracking entry tables, each principal tracking entry would include: (1) the tag of B (typically 22 bits), (2) a bit vector that acts as a directory to keep the principal and the replicated tracking entries coherent (e.g., 16 bits for a 16-tile CMP model), and (3) an ID that points to the tile that is currently hosting B (e.g., 4 bits for a 16-tile CMP model). On the other hand, a replicated tracking entry includes only B's tag and the ID to B's current host tile. In contrast, in case of a single TR table, both the principal and the replicated tracking entries would each encompass a tag, a bit vector, an ID, and an indicative bit to distinguish between the two types of entries (required for replacement purposes). Clearly, the bit vector added to each replicated entry becomes in this case redundant. Thus, splitting the TR table into RTR and PTR might be preferable for reducing storage overhead.

Assuming a 16-tile CMP where each tile encompasses 32 KB I/D L1 caches and a 512 KB L2 cache bank and assume PTR and RTR tables each with 8 K entries per tile, C-AMTE demonstrates a 12% increase of on-chip cache capacity. To illustrate how the area overhead of C-AMTE scales, Fig. 4 shows the storage requirements of C-AMTE under 16-tile, 32-tile, 64-tile, 128-tile, 256-tile, 512-tile, and 1024-tile platforms. The figure shows that C-AMTE with full-map bit vector (one bit for every core) for each principal tracking entry (C-AMTE(Full)) scales poorly especially after involving more than 64 cores on a single chip. Clearly, what makes C-AMTE non-scalable to a large number of tiles is the bit vector associated with each principal tracking entry. C-AMTE, however, need not incorporate full-map vectors. Similar to sparse directories [9] and SGI Origin style design [17], C-AMTE can involve more compact (coarse) vectors to improve upon the poor scalability at a moderate bandwidth increase. For instance, a bit vector can contain one bit for every four cores (C-AMTE(Comp4)), or one bit for every eight cores (PDA(Comp8)) and rely on a broadcast or multicast protocol to track replicated tracking entries.

3.5. Qualitative comparison with closely related designs

Two of the closely related location strategies are those proposed and utilized by PageNUCA [6] and R-NUCA [12]. Chaudhuri [6] suggested PageNUCA which employs data migration at page granularity. Access patterns of cores are dynamically monitored and pages are migrated to banks that minimize the access time for the sharing cores. To locate the migratory pages at the L2 space, each core maintains at the L1 level two tables (organized exactly as TLBs) that map the original physical frame number of an instruction or data page to the migrated frame number. These tables are referred to as iL1Map and dL1Map, respectively. Upon each L2 request, the appropriate table is looked up before routing the request to the correct L2 bank. An entry in the appropriate L1Map is loaded from another unified map table (L2Map) maintained at the L2 level when the corresponding page table entry is loaded in the TLB at the time of a TLB miss. On a migration, the new physical frame number of a page is sent to the sharing cores (with the help of a sharing vector maintained at a table referred to as PACT) so that they can update their L1Map tables appropriately.

Hardavellas et al. [12] proposed R-NUCA that also relies on OS. R-NUCA classifies cache accesses to either private, shared, or instructions. Private pages are placed at the local L2 banks of the requesting cores, shared at fixed address-interleaved on-chip locations, and instructions at non-overlapping fixed-center clusters of L2 banks. R-NUCA extends page table and TLB entries to distinguish between private and shared pages. A request to L2 after a miss at the L1-I cache is immediately classified as an instruction and a direct location is simply performed assuming a fixed-center cluster centered at the requesting core. On the other hand, a request to L2 after a miss at an L1-D cache is resolved during the virtual-to-physical translation. The corresponding TLB (or page table in case of a TLB miss) entry is examined to decide upon

Table 2
System parameters.

Component	Parameter
Cache line size	64 B
L1 I/D-cache size/Associativity	16 KB each/2way
L1 read penalty (on hit per tile)	1 cycle
L1 replacement policy	LRU
L2 cache size/Associativity	512 KB per L2 bank or 8 MB aggregate/16way
L2 bank access penalty	12 cycles
L2 replacement policy	LRU
Latency per NoC hop	3 cycles
Memory latency	300 cycles

whether the requested page is private or shared. Subsequently, the request is routed to either the local (if the page is private) or home (if the page is shared) L2 bank.

Clearly, both R-NUCA and PageNUCA employ *direct* location strategies similar to C-AMTE. However, two main things differentiate C-AMTE from them. First, R-NUCA and PageNUCA are page-granular schemes that involve OS while C-AMTE is a block-granular strategy that is fully transparent and does not involve OS. Second, R-NUCA and PageNUCA are not pure location strategies but rather placement and migration designs, respectively. R-NUCA changes the original mappings of blocks and PageNUCA migrates blocks from their original home banks. As such, they both require ways to locate the L2 cache blocks that do not reside anymore at their home banks. Their suggested location strategies are *specific* to their proposed mechanisms. In contrast, C-AMTE is a pure location strategy that is general (i.e., not specific to any cache scheme). C-AMTE enables any block-granular placement and migration schemes.

4. Quantitative evaluation

In this work we assume a baseline block-interleaved shared CMP cache organization. We study C-AMTE with an implementation of the DNUCA scheme [13,2]. We employ DNUCA on our tiled CMP architecture via allowing migration in vertical and horizontal directions seeking to reduce hit latencies. Each cache block is augmented by four 2-bit saturation counters in correspondence to the four plausible ways: north, south, west, and east. Once a counter saturates, its value is cleared and the block is migrated towards the indicated direction (i.e., promoted up, down, left, or right one tile upon the saturation of the north, south, west, or east counter, respectively). To locate cache blocks after migration, C-AMTE is utilized. We refer to this DNUCA implementation with C-AMTE being incorporated as DNUCA(C-AMTE).

To demonstrate the potential performance gain from C-AMTE we compare DNUCA(C-AMTE) against the baseline shared (S) scheme and three other DNUCA implementations that are only different in their location processes. First, we consider DNUCA with a broadcast location strategy. That is, queries to all tiles are sent upon every L2 request to locate the required block. We denote this implementation as DNUCA(B). Second, a 3-way cache-to-cache transfer strategy is employed similar to the one in [29]. This implementation is designated as DNUCA(3W). Lastly, we consider DNUCA with an ideal location strategy to set an upper bound for C-AMTE and see how close it draws to a perfect approach. The ideal strategy assumes that cores have oracle knowledge about the on-chip residences of blocks. Hence, every L2 request is routed directly to the correct L2 bank. We refer to such an implementation as DNUCA(Ideal).

4.1. Methodology

We present our results based on a detailed full system simulation using Virtutech's Simics 3.0.29 [27]. We use our own CMP cache modules fully developed in-house. We implement the XY-routing algorithm and accurately model congestion for all types of 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. The tiles are organized as a 4×4 grid connected by a 2D mesh network on-chip (NoC). A 3-cycle latency (in addition to the NoC congestion delay) per hop is incurred when a datum traverses through the mesh network [30, 29]. Each tile encompasses a switch, an aggregate 32 KB I/D L1 cache, a 512 KB L2 cache bank, and a tracking table (TR) with 16 K 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 [6]. Lastly, for coherence enforcement at the L1 cache level, a distributed in-cache MESI-based directory protocol is employed (fully verified and tested). Table 2 details our configuration's experimental parameters.

We use a mixture of multithreaded and multiprogramming workloads to study the five designs, S, DNUCA(B), DNUCA(3W), DNUCA(C-AMTE), and DNUCA(Ideal). For multithreaded workloads we use the commercial benchmark SpecJBB [25], five shared memory programs from the SPLASH-2 suite [28] (Ocean, Barnes, Lu, Radix, and FFT), and three applications from the PARSEC suite [3] (Bodytrack, Fluidanimate, and Swaptions). We composed multiprogramming workloads using the above considered SPLASH-2 benchmarks and five other applications from SPEC2006 [25] (Hmmer, Sphinx, Milc, Mcf, and Bzip2). Table 3 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, MIX3, and MIX4 is run for 20 billion user instructions.

4.2. Results

Fig. 5 demonstrates the average L2 access latency (AAL) of S, DNUCA(B), DNUCA(3W), DNUCA(C-AMTE), and DNUCA(Ideal) schemes normalized to S. The incurred latency per L2 access is defined depending on three scenarios. First, it can involve only the L2 access time. This happens when a hit occurs to a local L2 bank from a requesting core. Second, it can incorporate distance latency (computed in terms of the number of hops traversed between the requester and the target tiles and the observed NoC congestion delay) and the L2 access time. This occurs upon a hit to a remote L2 bank. Third, it can involve memory latency because of a miss on L2. DNUCA(C-AMTE) achieves AAL improvement over S by an average of 18.4%, and by as much as 34.4% for Radix. This makes DNUCA(C-AMTE) significantly close to DNUCA(Ideal) which accomplishes, in contrast, an average AAL improvement of 23%. DNUCA(C-AMTE) does not draw nearer to DNUCA(Ideal) because of two main reasons: (1) misses to TR tables by requester cores and (2) overhead to keep the principal and the replicated tracking entries coherent after blocks' migrations. Consequently, DNUCA(C-AMTE) generates a higher NoC traffic which causes more NoC delay and, subsequently, inferior AAL accomplishment. Table 4shows the

Table 3
Benchmark programs.

Name	Input
SPECjbb	Java HotSpot (TM) server VM v 1.5, 4 warehouses
Ocean	514 × 514 grid (16 threads)
Barnes	32 K particles (16 threads)
Lu	2048 × 2048 matrix (16 threads)
Radix	3M integers (16 threads)
Bodytrack	4 frames and 1 K particles (16 threads)
Fluidanimate	5 frames and 300 K particles (16 threads)
Swaptions	64 swaptions and 20 K simulations (16 threads)
MIX1	Hmmer (reference) (16 copies)
MIX2	Sphinx (reference) (16 copies)
MIX3	Barnes, Ocean, Radix, Lu, Milc (ref), Mcf (ref) Bzip2 (ref), and Hmmer (2 threads/copies each)
MIX4	Barnes, FFT (4M complex numbers), Lu, and Radix (4 threads each)

Table 4
Message-hops per 1 K instructions.

	S	DNUCA(B)	DNUCA(3W)	DNUCA(C-AMTE)	DNUCA(Ideal)
SPECjbb	5.3	87.5	5.7	4.8	2.4
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
Swaptions	4.8	64.3	6.6	7.2	3.2
Bodytrack	5.2	95	8.5	11.3	4.9
Fluidanimate	11.3	174.9	11.88	11.82	10.3
MIX1	35.5	573.8	37.3	37.6	27.4
MIX2	22.1	370.2	32.6	47	19
MIX3	11.6	168	16.4	14.9	10.3
MIX4	50.8	691.7	54.8	80	26

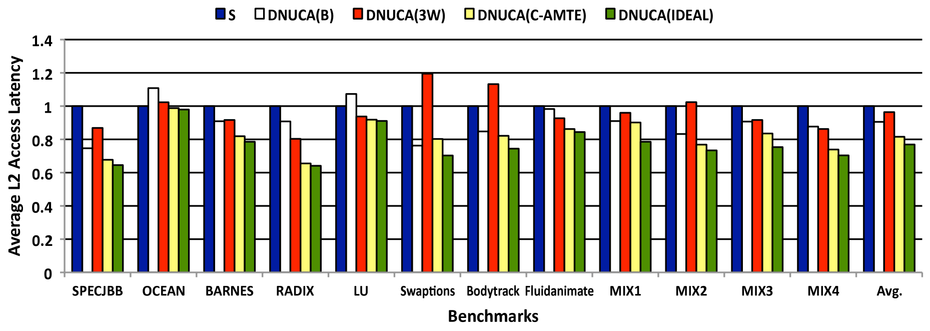


Fig. 5. Average L2 access latency of the baseline shared scheme (S), DNUCA(B), DNUCA(3W), DNUCA(C-AMTE), and DNUCA(Ideal) normalized to S (B = Broadcast, 3W = 3 Way).

number of message-hops per 1 K instructions experienced by all the studied schemes for the examined benchmark programs. A message-hop is defined as one message travelling one hop on a router in the 2D mesh NoC.

As shown in Fig. 5, DNUCA(B) and DNUCA(3W) provide AAL improvements over S by averages of 9.4% and 3.6%, respectively. DNUCA(B) is similar to DNUCA(Ideal) in that it offers a direct locations for cache blocks. However, DNUCA(B) profoundly outbursts the NoC with superfluous queries. This causes more NoC delay which translates to a lower AAL improvement. Two factors determine the eligibility of an application to accomplish a higher or a lower AAL under DNUCA(B): (1) the gain, *G*, out of direct locations to cache blocks and (2) the loss, *L*, from congestion delay. When *L* is offset by *G*, DNUCA(B) improves AAL (e.g., SpecJBB), otherwise, a degradation over S is observed (e.g., Ocean). DNUCA(3W), on the other hand, fails to exploit distance locality and is expected, accordingly, not to surpass S. Nonetheless, most of the applications experience AAL improvement under DNUCA(3W) (SpecJBB, Barnes, Radix, Lu, Fluidanimate, MIX1, MIX3, MIX4). This improvement comes, in fact, from the fewer off-chip accesses attained by DNUCA. Computer programs exhibit large asymmetry in cache sets' usages [21,19]. DNUCA inadvertently equalizes the non-uniformity across cache sets via the employment of the one-to-one associative mapping.

To that end, Fig. 6 presents the execution times of S, DNUCA(B), DNUCA(3W), DNUCA(C-AMTE), and DNUCA(Ideal) normalized to S. Across all benchmarks, DNUCA(B), DNUCA(3W), DNUCA(C-AMTE), and DNUCA(Ideal) outperform S by averages of 1.4%, 2.6%, 6.7%, and 8%, respectively. Though DNUCA(B) accomplished 9% and 9.2% AAL reductions over S for Barnes and Radix respectively, this did not effectively translate to an improvement in the overall system performance.

5. Concluding remarks and remaining work

Cache management in CMP is crucial to fuel its performance growth. This paper proposes C-AMTE, a mechanism that effectively simplifies the process of locating cache blocks in CMP caching schemes that employ either one-to-one or one-to-many associative mappings. C-AMTE stores tracking entries that correspond to cache blocks at per-core data structures for direct locations at subsequent accesses. We demonstrated the effectiveness of C-AMTE

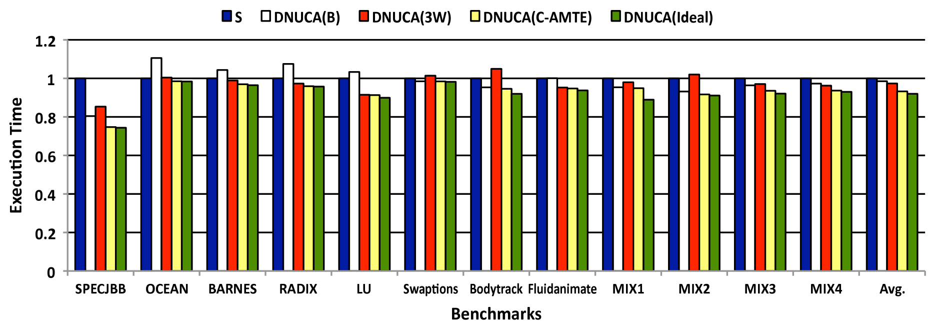


Fig. 6. Execution times of the baseline shared scheme (S), DNUCA(B), DNUCA(3W), DNUCA(C-AMTE), and DNUCA(Ideal) normalized to S (B = Broadcast, 3W = 3 Way).

by applying it to the DNUCA [2,13] migration scheme (i.e., a scheme that adopts one-to-one associative mapping). A performance improvement of up to 25.2% has been achieved, close to that of a perfect location strategy.

Lastly, having established the effectiveness of C-AMTE, optimizations to reduce hardware cost, a sensitivity study to different RT table sizes (or alternatively RTR and PTR tables), alternatives on evicting principal tracking entries, protocols on replacing blocks, and applying C-AMTE to more CMP caching schemes, specifically to schemes that incorporate one-to-many associative mapping (e.g., replication schemes), are among the obvious future directions.

References

[1] L. Barroso, et al. Piranha: a scalable architecture based on single-chip multiprocessing, in: ISCA, May 2000.
[2] B.M. Beckmann, D.A. Wood, Managing wire delay in large chip-multiprocessor caches, Micro (2004).
[3] C.M. Bienia, S. Kumar, J.P. Singh, K. Li, The PARSEC benchmark suite: characterization and architectural implications, PACT, October 2008.
[4] L. Censier, P. Feautrier, A new solution to coherence problems in multicache systems, IEEE Trans. Comput. C-27 (12) (1978) 1112–1118.
[5] J. Chang, G.S. Sohi, Cooperative caching for chip multiprocessors, in: ISCA, June 2006.
[6] M. Chaudhuri, PageNUCA: selected policies for page-grain locality management in large shared chip-multiprocessor caches, HPCA, February 2009, pp. 227–238.
[7] Z. Chishti, M.D. Powell, T.N. Vijaykumar, Optimizing replication, communication, and capacity allocation in CMPs, in: ISCA, June 2005.
[8] S. Cho, L. Jin, Managing distributed shared L2 caches through OS-level page allocation, Micro (2006).
[9] A. Gupta, W.D. Weber, T. Mowry, Reducing memory and traffic requirements for scalable directory-based cache coherence schemes, in: Int'l Conference on Parallel Processing, August 1990.
[10] Z. Guz, I. Keidar, A. Kolodny, U.C. Weiser, Utilizing shared data in chip multiprocessors. With the Nahalal architecture, in: SPAA, June 2008.
[11] M. Hammoud, S. Cho, R. Melhem, ACM: an efficient approach for managing shared caches in chip multiprocessors, in: HiPEAC, January 2009, pp. 319–330.
[12] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki, Reactive NUCA: near-optimal block placement and replication in distributed caches, in: ISCA, June 2009.
[13] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, S.W. Keckler, A NUCA substrate for flexible CMP cache sharing, in: ICS, June 2005.
[14] M. Kandemir, F. Li, M.J. Irwin, S.W. Son, A novel migration-based NUCA design for chip multiprocessors, SC, November 2008.
[15] C. Kim, D. Burger, S.W. Keckler, An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches, in: ASPLOS, October 2002, pp. 211–222.
[16] P. Kongetira, K. Aingaran, K. Olukotun, Niagara: a 32-way multithreaded SPARC processor, IEEE Micro 25 (2) (2005) 21–29.
[17] J. Laudon, D. Lenoski, The SGI origin: a ccNUMA highly scalable server, in: ISCA, June 1997.
[18] M.R. Marty, M.D. Hill, Virtual hierarchies to support server consolidation, in: ISCA, June 2007.
[19] M.K. Qureshi, D. Thompson, Y.N. Patt, The V-Way cache: demand-based associativity via global replacement, in: ISCA, June 2005, pp. 544–555.
[20] Research at Intel, Introducing the 45 nm Next-Generation Intel Core® Microarchitecture, White Paper.
[21] D. Rolán, B.B. Fraguela, R. Doallo, Adaptive line placement with the set balancing cache, Micro (2009) 529–540.
[22] K. Strandberg, Which OS? Considerations for performance-asymmetric, multi-core platforms, Research at Intel, White Paper.
[23] B. Sinharoy, R.N. Kalla, J.M. Tendler, R.J. Eickemeyer, J.B. Joyner, POWER5 system microarchitecture, IBM J. Res. Dev. 49 (1) (2005) 25.
[24] S. Srikantaiah, M. Kandemir, M.J. Irwin, Adaptive set pinning: managing shared caches in chip multiprocessors, in: ASPLOS, March 2008, pp. 135–144.
[25] Standard performance evaluation corporation. http://www.specbench.org.
[26] D. Tam, R. Azimi, L. Soares, M. Stumm, Managing shared L2 caches on multicore systems in software, in: WIOSCA, 2007.
[27] A.B. Virtutech, Simics full system simulator. http://www.simics.com/.
[28] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, A. Gupta, The SPLASH-2 programs: characterization and methodological considerations, in: ISCA, July 1995, pp. 24–36.
[29] M. Zhang, K. Asanović, Victim migration: dynamically adapting between private and shared CMP caches, TR-2005-064, MIT, October 2005.
[30] M. Zhang, K. Asanović, Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors, in: ISCA, June 2005.

Mohammad Hammoud is a graduate student in the computer science department at the University of Pittsburgh. Currently he is working under the supervision of Professor Rami Melhem and Professor Sangyeun Cho. In June 2004 he received his bachelor degree in computer science from the American University of Science and Technology (AUST), Lebanon. After that he was enrolled as a full time graduate student in the same school before transferring in August 2005 to the University of Pittsburgh.

Sangyeun Cho received his B.S. in Computer Engineering from Seoul National University, Seoul, in 1994, and his Ph.D. in Computer Science from the University of Minnesota, Minneapolis, in 2002.

From 1999 to 2004, he worked for Samsung Semiconductor, where he designed several generations of the CalmRISC(TM) embedded processor core and their cache memories. His research focus is in the area of computer architecture, microprocessor design, and system-on-a-chip (SOC).

Dr. Cho joined the Department of Computer Science at the University of Pittsburgh in fall 2004.

Rami Melhem has received the following degrees: B.S. (Electrical Engineering, 1976) from Cairo University; B.S. (Mathematics, 1978) from Ein-Shams University, Cairo; M.A. (Mathematics, 1981), M.S. (Computer Science, 1981), and Ph.D. (Computer Science, 1983) from the University of Pittsburgh.

He was Assistant Professor in the Department of Computer Science at Purdue University 1984–87 (on leave 1985–87), and Visiting Professor in the Department of Mathematics at the University of Pittsburgh 1985–86.

Since 1986 he has been on the faculty of the Department of Computer Science at the University of Pittsburgh. He has published numerous papers in the areas of systolic architectures, parallel computing, fault-tolerant computing, and optical interconnection networks. He served on program committees for several conferences and is on the Editorial Board of IEEE Transactions on Computers. He is a member of the IEEE Computer Society, the Association for Computing Machinery, and the International Society for Optical Engineering.

His research interests include: parallel and distributed high-performance computing, fault-tolerant computing, multiprocessor interconnection networks, real-time systems and optical computing.