A Novel Mapping Policy for Distributed Shared Caches

Alberto Ros, Manuel E. Acacio, José M. García
Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia
{a.ros,meacacio,jmgarcia}@ditec.um.es

Marcelo Cintra
School of Informatics
University of Edinburgh
mc@inf.ed.ac.uk

Abstract

In many-core architectures, memory blocks are commonly assigned to the banks of a distributed shared cache by following a physical mapping. This mapping assigns blocks to cache banks in a round-robin fashion without considering the distance between the cores that access a block and its corresponding cache bank. This fact increases both cache access latency and on-chip network traffic. On the other hand, first-touch mapping policies, which take into account distance, can lead to an unbalanced utilization of cache banks, thus increasing the number cache misses. In this work, we propose a mapping policy that addresses the trade-off between cache access latency and cache misses without requiring any extra hardware structure. We show that our proposal obtains average improvements of 11% for parallel applications and 14% for multi-programmed workloads in terms of execution time, and significant reductions in network traffic over a traditional physical mapping.

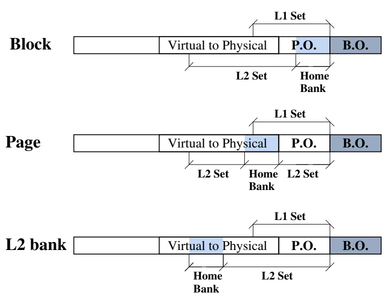
1 Introduction

Chip multiprocessors (CMP) are already a commercial reality (i.e., the 2-core IBM Power6 [10] and the 8-core Sun T2 [14]) and CMP architectures that integrate tens of processor cores (usually known as many-core CMPs) are expected for the near future [2]. Particularly, tiled CMP architectures, which are designed as arrays of identical or close-to-identical building blocks (tiles), are a scalable alternative to current small-scale CMP designs and will help in keeping complexity manageable. Each tile is mainly comprised by a core, one or several levels of caches, and a network interface that connects all tiles through a point-to-point network.

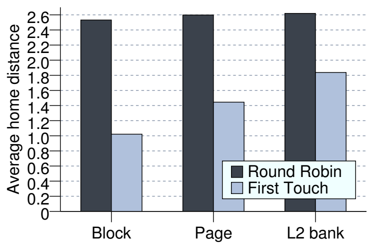
In tiled CMPs, the distributed last-level on-chip cache (the L2 cache in this work) can be logically shared among all cores or separated into private banks. The shared L2 cache organization, also called distributed shared cache or non-uniform cache architecture (NUCA) [9], achieves a better use of the L2 cache when compared to a private organization. This is because it only stores one copy of each block and it distributes the copies across the different banks. The main downside of this organization is the long L2 access latency, since it depends on the bank wherein the block is allocated, i.e., the *home* bank or tile. This issue is addressed in this work.

In a shared organization, the most straightforward way of distributing blocks among the different tiles is by using a physical mapping policy in which a set of bits in the block address defines the home bank for every block. Some recent proposals [8] and commercial CMPs [10, 14] choose the less significant bits[1] for selecting the home bank (*Block* diagram in Figure 1(a)). In this way, blocks are assigned to banks in a round-robin fashion with block-size granularity. This physical mapping uniformly spreads blocks among cache banks, resulting in optimal utilization of the cache storage. However, it does not take into account

[1] In this paper, when we refer to the less significant bits of an address we are not considering the block offset.



(a) Different granularities of interleaving (P.O.=Page offset, B.O.=Block offset).



(b) Impact on average home distance for the SPLASH-2 benchmark suite and 16 cores.

Figure 1: Granularity of L2 cache interleaving and its impact on average home distance.

the distance between the requesting core and the home bank on a L1 cache miss. Moreover, the average distance between two tiles significantly increases with the size of the CMP, which can become a performance problem for many-core CMPs.

An alternative mapping policy is *first-touch* [7], which has been widely used in NUMA architectures to achieve more locality in the memory accesses. First-touch minimizes the distance between the requesting cores and the home banks, thus reducing the cache access latency. However, an efficient implementation of this policy requires the OS to handle it, and therefore, the granularity of mapping must be at least the size of a page (e.g., *Page* or *L2 bank* diagram in Figure 1(a)). The OS maps a page to a particular cache bank the first time the page is referenced, i.e, a memory miss. At that moment, the OS assigns a physical address to the virtual address of the page, thus changing some bit in the address of the page (*Virtual to Physical* field in figure 1(a)). Then, the OS can control the cache mapping by assigning to this page a physical address that maps to the desired bank. A first-touch policy is easily implemented by assigning an address that physically maps to the tile wherein the core that is accessing the page resides.

Although whit a round-robin mapping policy the granularity of the interleaving does not significantly affect the average distance to the home bank, with a first-touch mapping policy, finer granularity offers shorter average distance between the missing L1 cache and the home L2 bank, as shown in Figure 1(b). Therefore, it is preferable to use a grain size as fine as possible. Since block granularity is not suitable for OS-managed mapping, the finest granularity possible is achieved by taking the less significant bits of the *Virtual to Physical* field, i.e., a page-grained interleaving.

The main drawback of a first-touch policy is that applications with a working set not balanced among cores do not make optimal use of the total L2 capacity. This happens more frequently in systems used for throughput computing [4], where different applications with different memory requirements run on the same system. The use of these architectures as commercial servers emphasize the need of efficient mapping policies.

In this work, we propose the *distance-aware round-robin* (DARR) mapping policy, an OS-managed policy which does not require extra hardware. This policy tries to map the pages to the local bank of the first requesting core, like a first-touch policy, but also introduces an upper bound on the deviation of the distribution of memory pages among cache banks, which lessens the number of off-chip accesses.

Our proposal obtains average improvements of 11% for parallel applications and 14% for multi-programmed workloads over a round-robin policy. In terms of network traffic, our proposal obtains average reductions of 39% for parallel applications and 65% for multi-programmed workloads. When compared to a first-touch policy average improvements of 5% for parallel applications and 6% for multi-programmed workloads are obtained, slightly increasing on-chip network traffic.

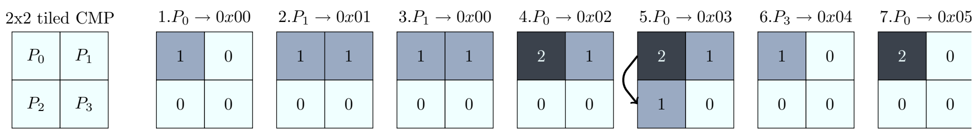


Figure 2: Behavior of the distance-aware round-robin mapping policy.

The rest of the paper is organized as follows. Section 2 describes the proposed mapping policy. Section 3 introduces the methodology employed. Section 4 shows the performance results. Section 5 presents the related work and, finally, Section 6 concludes the paper.

2 DARR Mapping Policy

In this work, we propose distance-aware round-robin mapping, a simple OS-managed mapping policy for many-core CMPs that assigns memory pages to NUCA cache banks. This policy minimizes the total number of off-chip accesses as happens with a round-robin mapping, and reduces the access latency to a NUCA cache (the L2 cache level) as a first-touch policy does. Moreover, this policy addresses this trade-off without requiring any extra hardware support.

In the proposed mechanism, the OS starts assigning physical addresses to the requested pages according to a first-touch policy, i.e, the physical address chosen by the OS maps to the tile of the core that is requesting the page. The OS stores a counter for each cache bank which is increased whenever a new physical page is assigned to this bank. In this way, banks with more physical pages assigned to them will have higher value for the counter.

To minimize the amount of off-chip accesses we define an upper bound on the deviation of the distribution of pages among cache banks. This upper bound can be controlled by the OS through a threshold value. In case that the counter of the bank where a page should map following a first-touch policy has reached the threshold value, the page is assigned to another bank. The algorithm starts checking the counters of the banks at one hop from the initial placement. The bank with smaller value is chosen. Otherwise, if all banks at one hop have reached the threshold value, then the banks at a distance of two hops are checked. This algorithm iterates until a bank whose value is under the threshold is found. The policy ensures that at least one of the banks has always a value smaller than the threshold value by decreasing by one unit all counters when all of them have values different than zero.

Figure 2 shows, from left to right, the behavior of this mapping policy for a 2×2 tiled CMP with a threshold value of two. First, processor P_0 accesses a block within page $0x00$ which faults in memory (1). Therefore, a physical address that maps to the bank 0 is chosen for the address translation of the page, and the value for the bank 0 is increased. Then, processor P_1 perform the same operation for page $0x01$ (2). When processor P_1 accesses page $0x00$ no action is required for our policy because there is a hit in the page table (3). The next access of processor P_0 is for a new page, which is also stored in bank 0, which reaches the threshold value (4). Then, if processor P_0 accesses a new page again, this page must be allocated in another bank (5). The closer bank with a smaller value is bank 2. Finally, when processor P_3 accesses a new page, the page is assigned to its local bank and all counters are decreased (6), allowing bank 0 to map a new page again (7).

The threshold defines the behavior of our policy. A threshold value of zero denotes a round-robin policy in which a uniform distribution of pages is guaranteed, while an unlimited threshold implies a first-touch policy. Therefore, with a small threshold value, our policy reduces the number of off-chip accesses. Otherwise, if the threshold value is high, our policy reduces the average latency of the accesses to the NUCA cache. Although, the OS could choose different thresholds depending on the workload, we have found that values between 64 and 256 work well for the workloads considered in this work.

Table 1: System parameters.

Memory Parameters: GEMS (4GHz)	
Cache block size	64 bytes
Split L1 I & D caches	64KB, 4-way
L1 cache hit time	3 cycles
Shared unified L2 cache	512KB/tile, 16-way
L2 cache hit time	6 cycles
Memory access time	300 cycles
Page size	4KB
Network Parameters: SICOSYS (2GHz)	
Topology	2-dimensional mesh
Switching technique	Wormhole
Routing technique	Deterministic X-Y
Data and control message size	4 flits and 1 flit
Routing / Switch time	1 cycle / 1 cycle
Link latency (one hop)	2 cycles
Link bandwidth	1 flit/cycle

3 Simulation Environment

We evaluate our proposal using the Simics full-system multiprocessor simulator [11] extended with GEMS [12] and SiCoSys [13]. GEMS provides a detailed cache coherent memory system timing model and SiCoSys simulates a detailed interconnection network that allows one to take into account most of the VLSI implementation details with high precision.

Besides the policy already provided by GEMS, a physical mapping with block-grained interleaving that we call *Block-RoundRobin*, we have implemented the other three OS-managed policies evaluated in this work. The first one, named as *Page-RoundRobin*, is an OS-managed policy that assigns physical pages in a round-robin fashion to guarantee the uniform distribution of pages. Therefore, this policy does not take into consideration the distance to the home bank. The second one, named as *Page-FirstTouch*, maps memory pages to the local cache bank of the first processor that requested the page. Although this policy is distance-aware, it is not concerned about the load on some cache banks. Finally, we also implement the policy proposed in this work. We simulate our proposal with threshold values ranging from 2^0 to 2^{10}. We call our policy *Page-DARR-T*, where *T* is the threshold value.

The simulated system is a tiled CMP in which each tile contains an in-order processor core. Table 1 shows the values for the main parameters of the system evaluated in this work. Memory blocks stored in the private L1 caches are kept coherent by means of a directory-based cache coherence protocol that uses MESI states.

3.1 Benchmarking

We have evaluated our proposal with parallel and multi-programmed workloads. Multi-programmed workloads consist of several program instances running at the same time in the system. We classify workloads as either *homogeneous* or *heterogeneous*. Homogeneous workloads uniformly distribute memory pages among cache banks when a first-touch policy is employed. In contrast, in heterogeneous workloads a few banks allocate more pages than the others considering a first-touch policy.

For evaluating the parallel applications we have chosen two homogeneous and two heterogeneous scientific benchmarks. *FFT* (256K complex doubles), with a small working set, and *Ocean* (258x258 ocean), with a larger working set, represent the homogeneous workloads. *Unstructured* (Mesh.2K, 5 time steps), with small working set, and *Radix* (1M keys, 1024 radix), with a larger working set, constitute the heterogeneous workloads. *FFT*, *Ocean* and *Radix* belong to the SPLASH-2 benchmark suite [15] while *Unstructured* is a computational fluid dynamics application. Since, in general, the working set of the scientific benchmarks is small we have shrunk the simulation parameters to 32KB 2-way L1 caches, 128KB 4-way L2 caches and 16 cores.

Since multi-programmed workloads have bigger working sets, we can fairly simulate a 32-core CMP with the cache sizes shown in Table 1. We have simulated two homogeneous and two heterogeneous workloads. *Ocean4* and *Radix4* consist of four instances of the *Ocean* and *Radix* applications, respectively, with eight threads each one, representing homogeneous workloads. *Mix4* and *Mix8* run *Ocean*, *Raytrace* (teapot), *Water-NSQ* (512 molecules, 4 time steps) and *Unstructured*. In *Mix4* each application has eight threads. In *Mix8* two instances of each application are run with four threads each. These two workloads represent the heterogeneous and more common multi-programmed workloads.

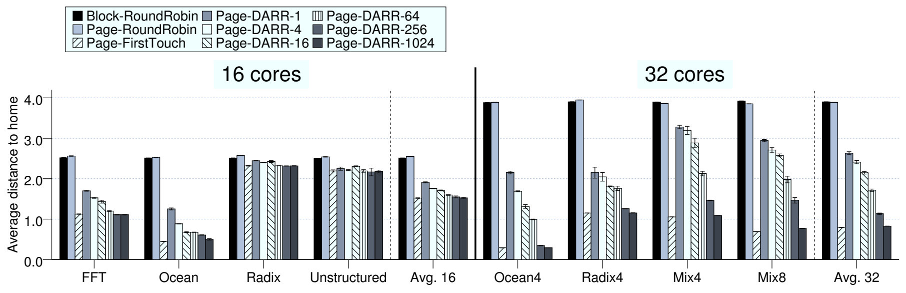


Figure 3: Average distance between requestor and home tile for the workloads evaluated in this work.

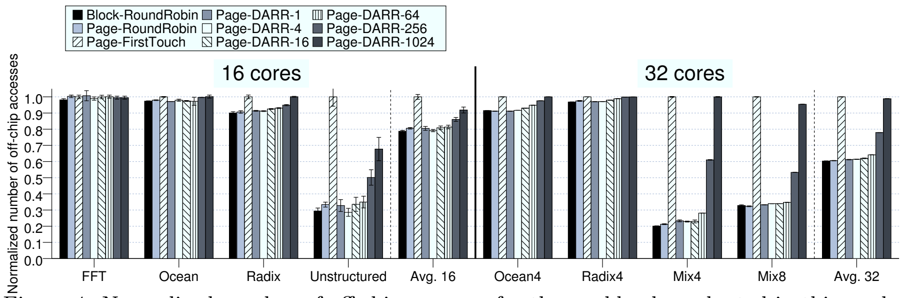


Figure 4: Normalized number of off-chip accesses for the workloads evaluated in this work.

4 Evaluation Results

4.1 Average distance to the home banks

Figure 3 plots the average distance in terms of network hops between the tile where the miss takes place and the tile where the home L2 bank is placed. As discussed, a round-robin policy does not care about this distance and, therefore, the average distance for these policies matches up with the average number of hops in a two-dimensional mesh (2.5 for a 4×4 mesh and 3.875 in a 4×8 mesh). On the other hand, the first-touch policy is the one that requires less hops to solve a miss (1.58 for parallel applications and 0.82 for multi-programmed workloads). As can be observed, the results obtained by our policy always lie between those of the round-robin and first-touch schemes.

Some parallel applications, like *Radix* and *Unstructured* do not obtain representative reductions in the average distance, even when a first-touch policy is considered. This is because the blocks that frequently cause a cache miss are widely shared. On the other hand, we can observe that the multi-programmed workloads always achieve important reductions in the average distance. Even when all the applications running in the system are instances of *Radix*, which does not offer reductions in the parallel case, as happens in *Radix4*. This is because data is only shared in the region of the chip running each instance.

Finally, it is important to note that a threshold value of one for our policy reduces the average distance compared to round-robin (by 25% for parallel and 32% for multi-programmed workloads), and also guarantees a uniform distribution of pages. The higher threshold value is employed, the more reductions in the average number of hops are achieved.

4.2 Number of off-chip accesses

The main issue of the first-touch policy is that it incurs in more off-chip accesses specially for workloads that have unbalanced working sets. Figure 4 shows the number of off-chip accesses for the implemented policies normalized with respect to *first-touch*. We can observe that for homogeneous workloads the difference in the number of off-chip accesses is minimal. On the other hand, the first-touch policy severely increases the number of off-chip accesses for het-

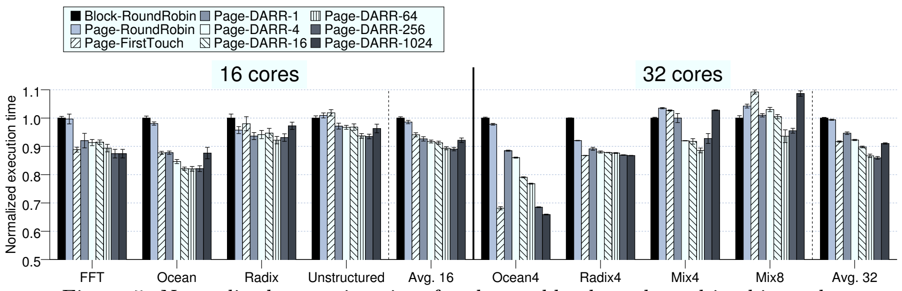


Figure 5: Normalized execution time for the workloads evaluated in this work.

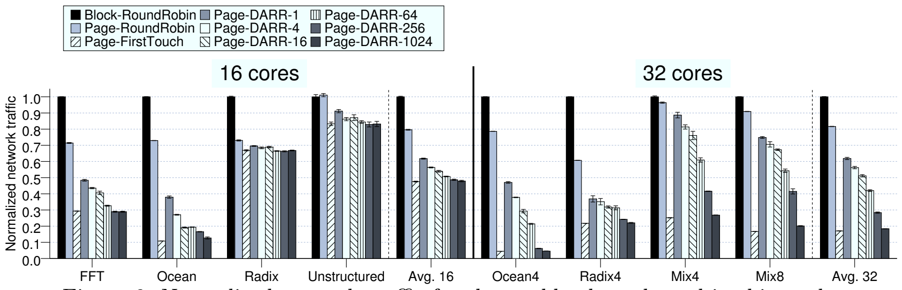


Figure 6: Normalized network traffic for the workloads evaluated in this work.

erogeneous workloads. This increment happens mainly in *Unstructured* ($\approx \times 3$), *Mix4* ($\approx \times 5$) and *Mix8* ($\approx \times 3$). Note that servers usually run a heterogeneous set of applications, like *Mix4* and *Mix8*.

Regarding the threshold value of our policy, we can observe that with a value smaller than 256 the number of off-chip accesses is kept very close to the round-robin policy.

4.3 Execution time

As we can observe in Figure 5, the first-touch policy achieves important improvements compared to a round-robin policy when the working set accessed by the different cores is homogeneous, as happens in *FFT*, *Ocean*, *Ocean4* and *Radix4*. In contrast, when the distribution of pages accessed by each core is heterogeneous, as occurs in *Radix*, *Unstructured*, *Mix4* and *Mix8*, first-touch incurs in more off-chip accesses, thus degrading performance. In contrast, our proposal achieves the best of a round-robin policy and a first-touch policy with a threshold value between 64 and 256. In this way, we obtain improvements of 11% on average for parallel applications and of 14% on average for multi-programmed workloads compared to a round-robin policy with page-sized granularity. When compared to a first-touch policy we obtain improvements of 5% for parallel applications and 6% for multi-programmed workloads, but additionally avoiding the performance degradation incurred by the first-touch policy in some cases.

4.4 Network traffic

Figure 6 compares the network traffic generated by the policies considered in this work. In particular, each bar plots the number of bytes transmitted through the interconnection network normalized with respect to *Block-RoundRobin*. We can see that round-robin policies lead to the highest traffic levels because the distance to the home bank is not taken into consideration.

On the other hand, network traffic can be tremendously reduced when a first-touch policy is implemented. In parallel applications, network traffic is reduced by 40% on average. For multi-programmed workloads the savings are greater (72% on average), since most of the blocks are only accessed by cores placed in a small region of the chip. Our policy always obtains reductions in network traffic compared to round-robin, even when the threshold value is just one. When the threshold value increases less network traffic is generated. We can see that with a threshold of 256 the network traffic generated by our proposal is reduced by 39% for parallel applications and 65% for multi-programmed workloads. Obviously, the first-touch policy introduces less traffic than our proposal (3% on average for parallel applications and 31% on average for multi-programmed workloads), at the cost of increasing the number of off-chip accesses.

5 Related Work

Kim *et al.*[9] presented non-uniform cache architecture (NUCA) caches. They studied both a static mapping of blocks to caches and a dynamic mapping based on *spread sets*. In such dynamic mapping, a block can only be allocated in a particular *bank set*, but this bank set can be comprised of several cache banks that act as *ways* of the bank set. In this way, a memory block can migrate from a bank far from the processor to another bank closer if the block is expected to be accessed frequently. Chishti *et al.* [6] achieved more flexibility than the original dynamic NUCA approach by decoupling tag and data arrays, and by adding some pointers from tags to data, and vice versa. Again, memory blocks can reside in different banks within the same bank set. Beckmann and Wood [3] proposed a new distribution of the components in the die, where the processing cores are placed around the perimeter of a NUCA L2 cache. Migration is also performed among cache banks belonging to the same bank set. The block search is performed in two phases, both requiring broadcasting the requests. Unfortunately, these proposals have two main drawbacks. First, there are data placement restrictions because data can only be allocated in a particular bank set and, second, data access requires checking multiple cache banks, which increases network traffic and power consumption.

Differently from all the previous approaches, and closer to ours, Cho and Jin [7] proposed using a page-size granularity (instead of block-size). In this way, the OS can manage the mapping policy, e.g, a first-touch mapping policy can be implemented. In order to deal with the unbalanced utilization of the cache banks, they propose using bloom filters that collect cache access statistics. If a cache bank is *pressured*, the neighbouring banks can be used to allocate new pages. For this proposal it is difficult to find an accurate metric to decide whether a cache is pressured or not. In contrast, in our proposal pages are distributed in an easy way and without requiring any extra hardware. Recently, Awasthi *et al.* [1] and Chaudhuri [5] proposed several mechanisms for page migration that reduce the overhead of migration at the cost of requiring extra hardware structures. Unfortunately, since migration of pages entails an inherent cost (e.g., flushing caches or TLBs), this mechanism cannot be performed frequently. Although migration can be used along with our proposal, this work focuses on the initial mapping of pages to cache banks.

6 Conclusions

In CMP architectures, memory blocks are commonly assigned to the banks of a NUCA cache by following a physical mapping policy, thus neglecting the distance between the requesting cores and the home NUCA bank for the requested blocks. This issue impacts both cache access latency and the amount of on-chip network traffic generated, and can become a performance problem for large-scale CMPs. On the other hand, first-touch mapping policies, which take into account distance, can lead to an unbalanced utilization of cache banks, and consequently, to an increased number of expensive off-chip accesses.

In this work, we propose the *distance-aware round-robin* mapping policy, an OS-managed policy which addresses the trade-off between cache access latency and number of off-chip accesses. Our policy tries to map the pages accessed by a core to its closest bank, like in a first-touch policy. However, we also introduce an upper bound on the deviation of the distribution of memory pages among cache banks, which lessens the number of off-chip accesses. This upper bound can be controlled by a threshold for which we have observed that our proposal achieves a good compromise between a round-robin and a first-touch policy with a value between 64 and 256.

Our proposal obtains average improvements of 11% for parallel applications and of 14% for multi-programmed workloads compared to a round-robin policy. In terms of network traffic, our proposal obtains average improvements of 39% for parallel applications and 65% for multi-programmed workloads. When compared to a first-touch policy we obtain average improvements of 5% for parallel applications and 6% for multi-programmed workloads, slightly increasing on-chip network traffic. Finally, one of the main assets of our proposal is its simplicity, because it does not require any extra hardware structure.

Acknowledgments

This work has been supported by European Comission funds under HiPEAC Network of Excellence and under grant "Consolider Ingenio-2010 CSD2006-00046". Alberto Ros is supported by a research grant from Spanish MEC under the FPU national plan (AP2004-3735).

References

[1] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter. Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In *15th HPCA*, pages 250–261, Feb. 2009.

[2] M. Azimi, N. Cherukuri, and D. N. Jayasimha, et al. Integration challenges and tradeoffs for tera-scale architectures. *Intel Technology Journal*, 11(3):173–184, Aug. 2007.

[3] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *37th MICRO*, pages 319–330, Dec. 2004.

[4] S. Chaudhry, P. Caprioli, S. Yip, and M. Tremblay. High-performance throughput computing. *IEEE Micro*, 25(3):32–45, May 2005.

[5] M. Chaudhuri. PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches. In *15th HPCA*, pages 227–238, Feb. 2009.

[6] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *36th MICRO*, pages 55–66, Dec. 2003.

[7] S. Cho and L. Jin. Managing distributed, shared L2 caches through OS-level page allocation. In *39th MICRO*, pages 455–465, Dec. 2006.

[8] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for flexible CMP cache sharing. In *19th ICS*, pages 31–40, June 2005.

[9] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th ASPLOS*, pages 211–222, Oct. 2002.

[10] H. Q. Le, W. J. Starke, and J. S. Fields, et al. IBM POWER6 microarchitecture. *IBM Journal of Research and Development*, 51(6):639–662, Nov. 2007.

[11] P. S. Magnusson, M. Christensson, and J. Eskilson, et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, Feb. 2002.

[12] M. M. Martin, D. J. Sorin, and B. M. Beckmann, et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 33(4):92–99, Sept. 2005.

[13] V. Puente, J. A. Gregorio, and R. Beivide. SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems. In *10th Euromicro-PDP*, pages 15–22, Jan. 2002.

[14] M. Shah, J. Barreh, and J. Brooks, et al. UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC. In *IEEE Asian Solid-State Circuits Conference*, pages 22–25, Nov. 2007.

[15] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *22nd ISCA*, pages 24–36, June 1995.