The Auction: Optimizing Banks Usage in Non-Uniform Cache Architectures

Javier Lira[1], Carlos Molina[1,2] and Antonio González[1,3]
[1] Dept. of Computer Architecture, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
[2] Dept. of Computer Engineering, Universitat Rovira i Virgili, 43007 Tarragona, Spain
[3] Intel Barcelona Research Center, Intel Labs - UPC, 08034 Barcelona, Spain
javier.lira@ac.upc.edu, carlos.molina@urv.net and antonio.gonzalez@intel.com

ABSTRACT

The growing influence of wire delay in cache design has meant that access latencies to last-level cache banks are no longer constant. Non-Uniform Cache Architectures (NUCAs) have been proposed to address this problem. Furthermore, an efficient last-level cache is crucial in chip multiprocessors (CMP) architectures to reduce requests to the off-chip memory, because of the significant speed gap between processor and memory and the limited memory bandwidth. Therefore, a *bank replacement policy* that efficiently manages the NUCA cache is desirable. However, the decentralized nature of NUCA has prevented previously proposed replacement policies from being effective in this kind of caches. As banks operate independently of each other, their replacement decisions are restricted to a single NUCA bank. We propose a novel mechanism based on the *bank replacement policy* for NUCA caches on CMP, called *The Auction*. This mechanism enables the replacement decisions taken in a single bank to be *spread* to the whole NUCA cache. Thus, *global* replacement policies that rely on the current state of the NUCA cache, such as evicting the least frequently accessed data in the whole NUCA cache, are now feasible. Moreover, *The Auction* adapts to current program behaviour in order to relocate a line that is being evicted from a bank in the NUCA cache to the most suitable position in the whole cache. We propose, implement and evaluate three approaches of *The Auction* mechanism. We also show that *The Auction* manages the cache efficiently and significantly reduces the requests to the off-chip memory by increasing the hit ratio in the NUCA cache. This translates into an average IPC improvement of 8%, and reduces energy consumed by the memory system by 4%.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: System architectures

ICS'10, June 2–4, 2010, Tsukuba, Ibaraki, Japan.

General Terms

Design, Experimentation, Management, Performance

Keywords

Chip Multiprocessors (CMP), Non-Uniform Cache Architecture (NUCA), Bank Replacement Policy

1. INTRODUCTION

Constant advances in integrated circuit technology offer opportunities for microarchitectural innovation and have boosted microprocessor performance growth in recent decades. In the 1990s, the main strategy for dealing with each increase in integration density was to increase the clock rate and introduce microarchitectural innovations to exploit Instruction-Level Parallelism (ILP) in applications. However, fundamental circuit limitations, limited amounts of Instruction-Level Parallelism and the almost unaffordable power consumption of microprocessors led to the search for a more efficient use of silicon resources: chip multiprocessors (CMPs). This architecture consists of simpler processors that can work at a much lower clock rate, alleviating the power-consumption constraint, and assuming Thread-level parallelism (TLP) enables CMPs to take advantage of existing parallel applications. Server high-end applications, therefore, benefit the most from these platforms. Similarly, it is also expected that future desktop applications for recognition, mining and analysis will require a high number of cores [13]. At present, the main processor vendors have focused on this architecture, meaning that several CMPs [23, 24, 29], consisting of up to eight processors, are commercially available. Existing roadmaps and research trends such as the Intel Tera-scale [33] processor, however, show that the number of cores is going to increase in the future.

For any multiprocessor, the memory system is a pivotal component which can boost or decrease performance dramatically. CMP architecture typically incorporates large and complex cache hierarchies. Recent studies have proposed mechanisms for dealing with the new challenges to the memory system posed by CMP architectures, some of the most notable of these being cooperative caching [9, 10], victim replication [35], adaptive selective replication [6] and other works that exploit the private/shared cache partitioning scheme [14]. However, the increasing influence of wire delay in cache design means that access latencies to the last-level cache banks are no longer constant [3, 28]. Non-Uniform Cache Architectures (NUCAs) have been proposed [22] to address this problem. A NUCA divides the whole

cache memory into smaller banks and allows nearer cache banks to have lower access latencies than farther banks, thus mitigating the effects of the cache's internal wires. Therefore, each bank behaves as a regular cache and all of them are connected by means of an interconnection network.

Recent studies have explored mechanisms for placement, migration and access in NUCA caches on CMP architectures [4, 12, 17, 18, 19, 22, 30]. However, most previous research has ignored *bank replacement policy* or has adopted a replacement scheme that was originally designed for uniprocessors/uniform-caches.

The decentralized nature of NUCA prevents the replacement policies proposed in the literature from being effective for this kind of caches. As banks operate independently from each other, their replacement decisions are restricted to a single NUCA bank. In this paper, we propose a novel adaptive mechanism focused on *bank replacement policy* in CMP-NUCA architectures that we call *The Auction*. This mechanism enables the replacement decisions taken in a single bank to be *spread* to the whole NUCA cache. Thus, *global* replacement policies that rely on current state of the NUCA cache, such as evicting the least frequently accessed data in the whole NUCA cache, are now feasible.

The Auction starts when a replacement occurs in a bank in the NUCA cache. Then, it finds the best destination bank[1] to which the evicted data should be relocated, instead of evicting replaced data permanently from the NUCA cache or relocating it to a statically defined bank. Thus, *The Auction* is a flexible, implementable and affordable mechanism that can be adapted by defining the behaviour of the three participants that take part in it: *the owner*, *the controller* and *the bidders*. Section 4 describes this mechanism in further detail.

The remainder of this paper is structured as follows. Section 2 describes the baseline architecture assumed in our studies. Section 3 lays out the motivation for the proposed mechanism. Section 4 describes *The Auction* in further detail. Section 5 presents the experimental methodology, followed by the results in Section 6. Section 7 proposes and analyses two enhanced auction approaches. Related work is discussed in Section 8, and concluding remarks are given in Section 9.

2. BASELINE ARCHITECTURE

As shown in Figure 1, the baseline architecture consists of an eight-processor CMP based on that of Beckmann and Wood [7]. The processors are located on the edges of the NUCA cache, which occupies the central part of the chip. Each processor provides the first-level cache memory, composed of two separate caches: one for instructions and one for data. The NUCA cache is the second-level cache memory and is shared by the eight processors. The NUCA cache is divided into 256 banks structured in a 16x16 mesh that are connected via a 2D mesh interconnection network.

In general, a NUCA model can be characterized by describing how it behaves with the following four policies: *bank placement policy*, *bank migration policy*, *bank replacement policy* and *bank access policy*. Bank placement policy determines the banks in which data blocks can be mapped into the NUCA cache. Migration policy determines whether data movements are allowed after producing a hit in the NUCA cache to minimize cache access latency in future accesses. Bank replacement policy determines the final destination of the evicted data blocks. Last but not least, bank access policy determines the data searching algorithm in the NUCA cache memory space.

[1]Based on the decisions of the implemented approach of the auction.

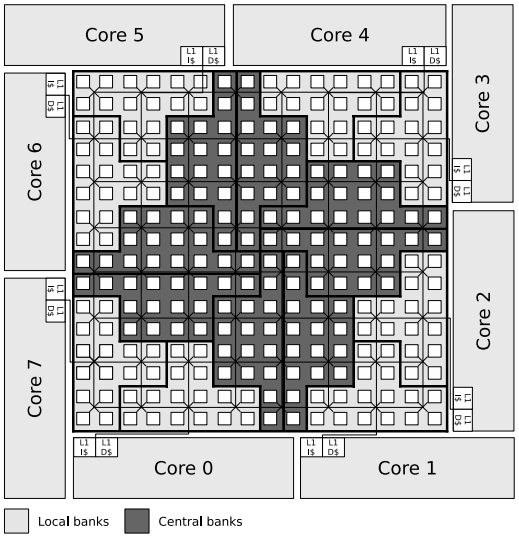


Figure 1: Baseline architecture layout.

As baseline, we implemented *Dynamic NUCA* (D-NUCA) [7, 22]. This architecture introduces a complex placement policy in which data blocks can be mapped into multiple banks within the NUCA cache. The 256 banks that compose the NUCA cache are *logically* separated into 16 unique banksets, where an address maps to a bankset and can reside within any bank of the bankset. Besides, D-NUCA *physically* separates the cache banks into 16 different bankclusters, shown as the shaded irregular pieces in Figure 1. Each bankcluster contains one bank from every bankset. The bankclusters are grouped into two distinct regions according to their physical distance from the processors. The 8 bankclusters closest to each processor form the *local banks* (lightly shaded in Figure 1). The other 8 bankclusters that reside in the center of the shared cache form the *central banks* (shaded dark in Figure 1). Therefore, a data block has *16 possible placements in the NUCA cache* (eight *local banks* and eight *central banks*). As migration policy, D-NUCA adopts *gradual promotion* [22]. When there is a hit in a NUCA bank, the accessed data is promoted to the bankcluster that is one-step closer to the processor that has just accessed it. With regard to the *bank access policy*, the baseline D-NUCA design uses a two-phase multicast search, that is also known as *partitioned multicast* [22]. First, it broadcasts a request to the *local* bankcluster that is closest to the processor that launched the search, and to the eight *central banks*. If all nine initial requests miss, the request is broadcast to the remaining seven banks from the requested data's bankset. If the request misses all 16 banks, it is sent to the off-chip memory. Finally, regarding *bank replacement policy*, D-NUCA assumes LRU replacement policy within each single bank, and sending the evicted data block directly to the off-chip memory (*zero-copy* [22]).

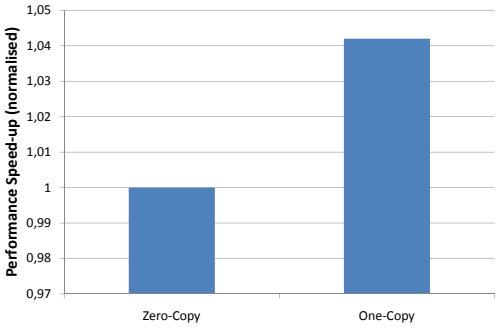


Figure 2: Speed-up assuming one-copy and zero-copy replacement policies.

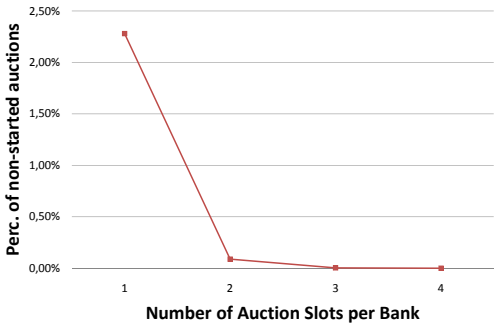


Figure 3: Percentage of non-started auctions when using up to four auction slots per NUCA bank.

3. MOTIVATION

The NUCA cache provides mechanisms to migrate accessed lines and take them closer to the core that requested them. Consequently, the most frequently accessed lines are stored in the banks that are closer to the cores, which we call *hot banks*. A replacement in a *hot bank*, however, evicts a line whose probabilities of being accessed farther in the program are much higher than a line from another bank in the NUCA cache. Moreover, as banks in the NUCA cache work independently of each other, none of the less used banks can even know that a *hot bank* is constantly evicting data that is being reused. Thus, a more sophisticated *bank replacement policy* that allows all banks in the NUCA cache to take part in data-replacement decisions is desirable so that lines evicted from the *hot banks* can be relocated to other banks in the NUCA cache, instead of being evicted them from the NUCA cache permanently. Unfortunately, most previous works have ignored the replacement issue or have adopted a replacement scheme that was originally designed for use in uniprocessors/uniform-caches.

Kim et al. [22] proposed two bank replacement policies for NUCA caches in a uniprocessor environment: *zero-copy* and *one-copy* policies. The evicted line in the NUCA cache, assuming the *zero-copy* policy, is sent back to the upper level of the memory hierarchy (the main memory in our studies). This is the policy implemented in the baseline architecture. If the *one-copy* policy is adopted, however, the evicted line is demoted to a more distant bank. This policy gives a second chance to the evicted lines to stay within the NUCA cache. In order to evaluate these schemes, we have adapted them for CMP. This version of the *one-copy* policy for CMP gives a second chance to the evicted lines by randomly relocating them to a bank from the bankset where they can be mapped.

Figure 2 shows that, the *one-copy* replacement policy improves performance compared to the baseline configuration[2]. *One-copy*, however, is considered as a *blind* replacement policy, insofar as it does not take into account the current cache state before relocating the evicted data to other NUCA bank. Thus, this approach may cause unfair data replacements that hurt performance.

In this paper we propose *The Auction* as the first bank replacement policy that fits the decentralized nature of CMP-NUCA architectures. This mechanism provides a protocol to *spread* replacement decisions that have been taken in a single bank to all banks in the NUCA cache. Thus, the other banks can take part in deciding which is the most appropriate data to evict within the whole NUCA cache. The following section describes *The Auction* mechanism.

[2]The experimental methodology is described in Section 5.

4. THE AUCTION

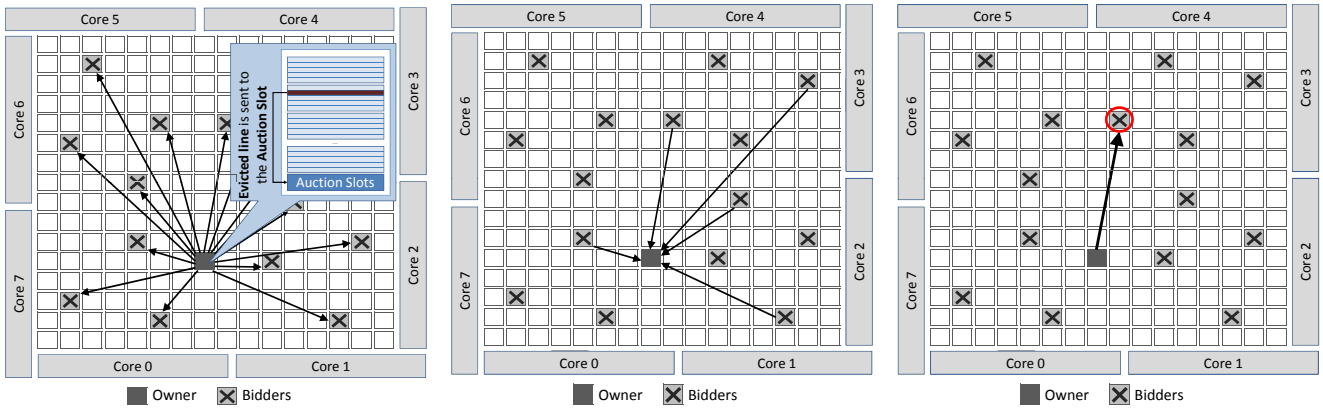
The Auction is an adaptive mechanism designed for the *bank replacement policy* of NUCA architectures in CMP. It provides a framework for *globalizing* the replacement decisions taken in a single bank, and thus enables the replacement policy to evict the most appropriate data from the NUCA cache. Moreover, unlike the *one-copy* policy (described in the previous section), The Auction enables evicted data from a NUCA bank to be relocated to the most suitable destination bank at any particular moment, taking into consideration the current load of each bank in the NUCA cache. This section describes in detail how *the auction* mechanism works.

4.1 Roles and components

In order to explain how *the auction* works, we will first introduce the three roles that operate in the mechanism:

- **Owner:** It owns the item but wants to sell it, thus starting the auction. The bank in the NUCA cache that evicts the line then acts as the *owner* and the evicted line is the *item* to sell.
- **Bidders:** They can bid for the item that is being sold in the auction. In the NUCA architecture, the bidders are the banks in the NUCA cache where the evicted line can be mapped. They are the other NUCA banks from the *owner's* bankset.
- **Controller:** It stores the item while the auction is running, receives the bids for the item from the bidders and manages the auction in order to sell the item to the highest bidder.

As *auction controller*, we introduce a set of *auction slots* that is distributed among all banks in the NUCA cache. Each auction slot manages a single active auction by storing the evicted line that is being sold, the current highest bidder and the remaining time. When the auction finishes the corresponding auction slot is deallocated and becomes available for forthcoming auctions. Therefore, the number of active auctions per NUCA bank is limited by the number of auction slots that it has. Figure 3 shows the percentage of



(a) Owner starts the auction (b) Bids for the item (c) Item is sold!

Figure 4: The three steps of the auction mechanism.

auctions that cannot be started because there are no auction slots available when using up to four auction slots per NUCA bank. We observe that assuming one auction slot per bank, just 2.3% of evicted lines can not be relocated. Moreover, we find that using more auction slots per NUCA bank, the percentage of non-started auctions dramatically decreases. At this point, the challenge is to determine the optimal number of auction slots that provides high accuracy without introducing prohibitive hardware overhead. In the remainder of the paper, we assume having *two auction slots per NUCA bank*. This configuration provides a good trade-off between auction accuracy and hardware requirements.

4.2 How The Auction works

Figure 4 shows the three steps of *the auction*. It starts when there is a replacement in a bank in the NUCA cache and it has at least one auction slot available – otherwise the auction can not be started and the evicted line is directly sent to the main memory –. The bank that is replacing data (the *owner*) moves the evicted line (the *item*) to the *controller* (the corresponding auction slot) and sets the auction deadline (Figure 4(a)). At the same time, the *owner* invites the other banks from the bankset (the *bidders*) to join the auction and bid for the *item*. Recall that an address maps to a bankset and can reside within any bank of the bankset. Thus, in our baseline architecture the evicted line can only be mapped to 16 banks within the whole NUCA cache. When a *bidder* finds out that a data block has been evicted from the NUCA cache, it decides whether to bid for it (Figure 4(b)). If the *bidder* is interested in getting the evicted data, it notifies the *controller* who manages *the auction*. Otherwise, the *bidder* ignores the current auction. Finally, when the auction time expires (Figure 4(c)), the *controller* determines the final destination of the evicted data based on the received bids and the implemented *heuristic*, and sends it to the winning *bidder*. Moreover, in order to avoid recursively starting auctions, even if relocating the evicted data provokes a replacement in the winning bank, it will not start a new auction. In contrast, if none of the *bidders* bid for the evicted data when the auction time expires, the *controller* sends it to the main memory.

Note that *The Auction* describes how to proceed when a replacement occurs in a bank in the NUCA cache. This is, therefore, a generic algorithm that must be customized by defining the decisions that each role can take on during the auction.

4.3 Implementing an Auction Approach

Most commonly used cache coherence protocols in multiprocessors (such as MESI, MSI and MOESI) allow data to be replicated when it is being accessed by several cores in read-only mode. Thus, these copies are stored in the cache in *shared* state. However, if one of the cores is going to update data, all the copies of this data that are in the cache have to be invalidated. This usually occurs in the highest-level cache that is not shared among cores (the L1 cache, in our architecture). However, NUCA features mean that it can also be in the shared-NUCA cache. As data can be mapped in several banks within the NUCA, data replication is allowed to provide low access latency to the cores that are accessing the same data.

The implemented auction approach takes advantage of the gaps in the NUCA banks provoked by data invalidation. First, the *owner* invites all the other banks from the bankset to join the current auction. When a *bidder* receives the auction notification from the owner, it checks whether there is an empty slot in the cache set where the evicted data would be mapped. If there is, the current bank bids for the evicted data. With regard to the *controller*, it will send the *item* to the NUCA bank whose bid arrived first to the controller, but prioritising *central banks*. As described in Section 3, migration movements make most frequently accessed data to be stored in *local banks*, thus if the *controller* receives bids from both types of banks, local and central, it will always prefer to send the item to the central bank. Then, the *controller* works as follows: If the first bid that arrives to the *controller* comes from a central bank, the auction finishes and the *item* is directly sent to the *bidder*. If the first *bidder* is a local bank, however, the *controller* sets the auction deadline to 20 cycles and waits for other bids from central banks. We have experimentally observed that even with high network contention most bids arrive to the

controller in 20 cycles from the arrival of the first bid. For the sake of clarity, in the remainder of the paper we call this auction approach *AUC-BASE*.

4.4 Hardware Implementation and Overhead

As described in Section 4.1, the auction mechanism introduces two *auction slots* per NUCA bank in order to manage the active auctions. Each auction slot requires 66 bytes (64 bytes to store the evicted line, 1 byte to identify the current highest bidder and 1 byte to determine the remaining time), the hardware overhead of this configuration is 33 KBytes (which is less than 0.4% of the total hardware used by the NUCA cache). Apart from hardware overheads, the auction also introduces extra messages into the on-chip network (i.e. messages to join the auction and bids). However, the impact of these messages on performance and energy consumption is taken into account by the simulator, and properly discussed in Section 6.

5. EXPERIMENTAL METHODOLOGY

We use the full-system execution-driven simulator, Simics [25], extended with the GEMS toolset [27]. GEMS provides a detailed memory-system timing model that enabled us to model the NUCA cache architecture. Furthermore, it accurately models the network contention introduced by the simulated mechanisms. The simulated architecture is structured as a single CMP made up of eight UltraSPARC IIIi homogeneous cores. With regard to the memory hierarchy, each core provides a split first-level cache (data and instructions). The second level of the memory hierarchy is the NUCA cache. We used the MOESI token-based coherence protocol [26] to maintain correctness and robustness in the memory system. Table 1 summarizes the configuration parameters used in our studies. The access latencies of the memory components are based on the models made with the CACTI 6.0 [31] modeling tool, this being the first version of CACTI that enables NUCA caches to be modeled.

Processors	8 - UltraSPARC IIIi
Frequency	1.5 GHz
Integration Technology	45 nm
Block size	64 bytes
L1 Cache (Instr./Data)	32 KBytes, 2-way
L2 Cache (NUCA)	8 MBytes, 256 Banks
NUCA Bank	32 KBytes, 8-way
L1 Latency	3 cycles
NUCA Bank Latency	4 cycles
Router Latency	1 cycle
Avg Offchip Latency	250 cycles
Auction time-out	150 cycles

Table 1: Configuration parameters.

In order to evaluate this mechanism, we assume two different scenarios: 1) Multi-programmed, and 2) Parallel applications. The former executes in parallel a set of eight different SPEC CPU2006 [2] workloads with the *reference* input. Table 2 outlines the workloads that make up this scenario. The latter, simulates the whole set of applications from the PARSEC v2.0 benchmark suite [8] with the *simlarge* input data sets. This suite contains 13 programs from many different areas such as image processing, financial analytics, video encoding, computer vision and animation physics, among others.

astar	gcc	lbm	mcf
milc	omnetpp	perlbench	soplex
Reference input			

Table 2: Set of SPEC CPU 2006 workloads that make up the multi-programmed scenario.

The method we used for the simulations involved first skipping both the initialization and thread creation phases, and then fast-forwarding while warming all caches for 500 million cycles. Finally, we performed a detailed simulation for 500 million cycles. As performance metric, we use the aggregate number of user instructions committed per cycle, which is proportional to the overall system throughput [34].

5.1 Energy Model

In this paper we evaluate the energy consumed by the NUCA cache and the off-chip memory. To do so, we used a similar energy model to that adopted by Bardine et al. [5]. This allowed us to also consider the static and dynamic energy dissipated by the NUCA cache and the additional energy required to access the off-chip memory. The total energy consumed by the memory system is the sum of all three components:

$$E_{total} = E_{static} + E_{dynamic} + E_{off-chip}$$

The NUCA cache was modeled using the CACTI 6.0 tool [31] to obtain the static energy (E_{static}). The dynamic energy ($E_{dynamic}$) consumed by the NUCA cache within the chip was modeled with the GEMS toolset [27] which uses the Orion simulator to determine the energy per bank access, the energy required to transmit a flit on the network link and the energy required to switch a flit through a network switch. The extra network traffic introduced by our proposal is also taken into account and accurately modeled into the simulator.

The energy dissipated per each access to the off-chip memory ($E_{off-chip}$) was determined using the *Micron System Power Calculator* [1] assuming a modern DDR3 system (4GB, 8DQs, Vdd:1.5v, 333 MHz). Our evaluation of the off-chip memory focuses on the energy dissipated during active cycles and isolates this from the background energy. This study shows that the average energy of each access is 550 pJ.

As energy metric we used the energy consumed per each memory access. It is based on the energy per instruction (EPI) [15] metric which is commonly used for analysing the energy consumption results of the whole processor. This metric works independently of the amount of time required to process an instruction and is ideal for throughput performance.

6. RESULTS AND ANALYSIS

This section analyses the impact on performance and energy consumption of using The Auction (AUC-BASE) as *bank replacement policy* in the baseline architecture. Unfortunately, none of the mechanisms previously proposed for NUCA caches on CMPs properly addresses the *bank replacement policy*, and thus they could complement the improvements achieved by *The Auction*. With regard to this policy, as we mention in Section 3, Kim et al. [22] proposed two different approaches, *zero-copy* and *one-copy*. However,

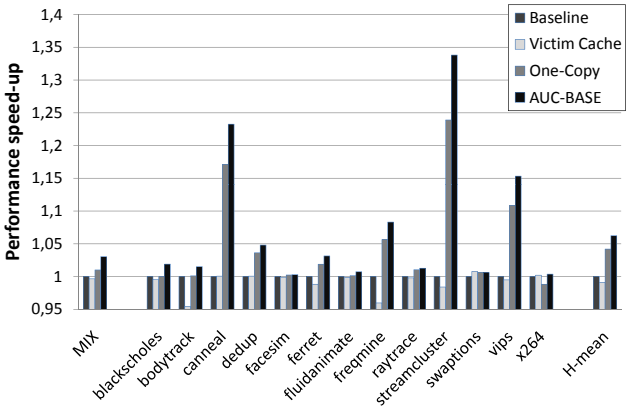


Figure 5: Performance improvement.

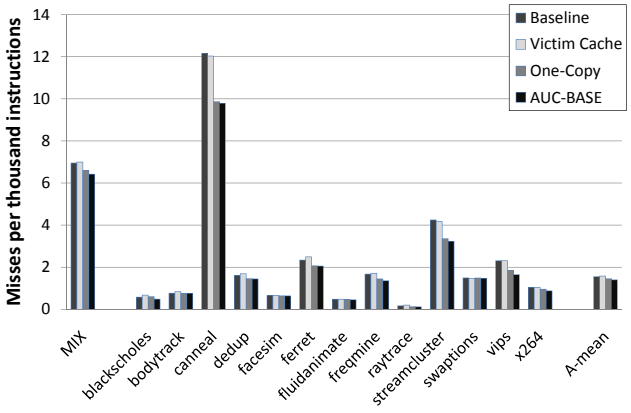


Figure 6: Misses per 1000 instructions.

these alternatives were proposed in a single-processor environment. Therefore, in order to compare them with the auction we have adapted them to the CMP baseline architecture. Moreover, we evaluate the baseline architecture with an extra bank that acts as a victim cache [20]. This mechanism does not show performance improvements on its own, however, it does introduce the same additional hardware as the auction approach.

6.1 Performance analysis

Figure 5 shows the performance improvement achieved when using *The Auction* for the *bank replacement policy* in the NUCA cache. On average, we find that AUC-BASE approach increases IPC by 6% compared to the baseline architecture. In general, we observe the auction performs significantly well with most of PARSEC applications, three of them improving IPC by more than 15% (*canneal*, *streamcluster* and *vips*). On the other hand, assuming the multi-programmed environment (*MIX* in Figure 5), the auction approach improves IPC by, on average, 4%.

One-copy replacement policy always relocates evicted data without taking into consideration the current state of the NUCA cache. This enables one-copy to improve performance in those PARSEC applications with large working sets, such as *canneal*, *streamcluster* and *vips*. However, blindly relocating evicted data could be harmful in terms of performance: for example, if *x264* is used, one-copy has a 2% performance loss. The Auction, on the other hand, checks the current state of all NUCA banks from the bankset where the evicted data can be mapped, and thus do not relocate evicted data if no a suitable destination bank has been found (i.e. if there are no bidders). This makes the auction a harmless mechanism in terms of performance even for applications with small working sets, such as *blackscholes* and *x264*. Moreover, we have experimentally observed that the performance benefits achieved using one-copy rely on the NUCA cache size, whereas the auction approaches do not correlate with the size of the cache.

Figure 5 shows that, on average, AUC-BASE increases IPC by 2% compared to one-copy. However, note that as replacement policy, the auction takes advantage of workloads with large working sets because they lead to more data replacements. For example, the auction increases IPC by 10% compared to one-copy using *streamcluster*, with *canneal* it increases IPC by 8%, and with *vips* by 5%. Unfortunately, most of PARSEC applications have small-to-medium working sets, and thus the average performance benefits achieved with a replacement policy are restricted.

Figure 6 shows the NUCA misses per 1000 instructions (MPKI) with the implemented auction approach (AUC-BASE). On average, we observe that there is a significant reduction in MPKI by using the auction. In general, we find that PARSEC applications that improve performance, also significantly reduce MPKI. Moreover, we should emphasize the fact that *canneal*, *streamcluster* and *vips* are the applications that both provide the highest IPC improvement with the auction and have the highest MPKI. In contrast, applications that have MPKI close to zero do usually not significantly improve performance when using this mechanism. On the other hand, in the multi-programmed environment, we find that the performance improvement is also related to a MPKI reduction.

6.2 Energy consumption analysis

In order to analyse the on-chip network contention introduced by the auction approach, Figure 7 shows the traffic on the on-chip network normalised to the baseline configuration. On average, both, one-copy and the auction, increases on-chip network traffic by 6%. Although both replacement mechanisms relocate evicted data, they also reduce miss rate in the NUCA cache compared to baseline configuration, so the increasing on the on-chip network traffic is not as high as previously expected. On the other hand, the auction also introduces extra messages into the on-chip network (auction invitations and bids). Figure 7 shows that the auction messages represents less than 10% of total on-chip network traffic from the AUC-BASE approach.

With regard to the energy consumption, Figure 8 shows that, on average, the auction reduces the energy consumed per each memory access by 4% compared to the baseline architecture. In particular, they significantly reduce energy consumption in PARSEC applications with large working sets, such as *canneal*, *freqmine*, *streamcluster* and *vips*. Regarding the multi-programmed environment, we also find similar results to multi-threaded applications in terms of energy consumption (4% reduction).

We conclude that as a replacement policy, this mechanism takes advantage of workloads with the largest working sets

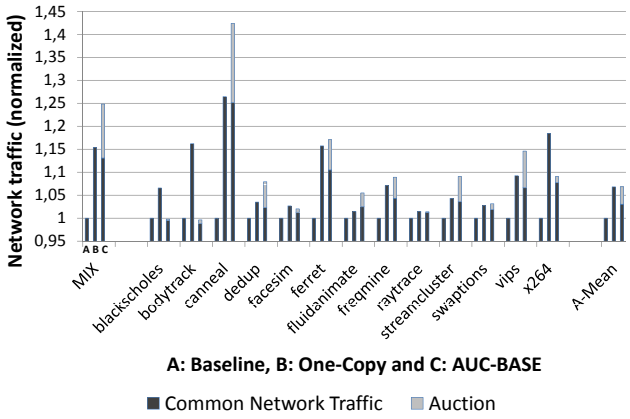


Figure 7: Network traffic.

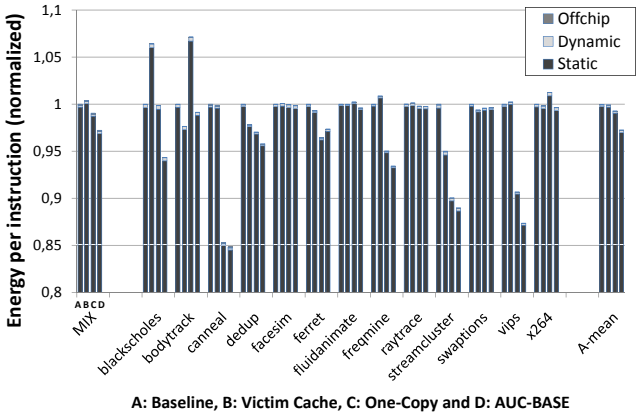


Figure 8: Energy per memory access.

because they lead to more data replacements and launch more auctions. On the other hand, we find that blindly relocating data in the NUCA cache without taking into consideration the current state of the banks, as is the case with one-copy, may cause unfair data replacements that can hurt performance.

7. ENHANCED AUCTION APPROACHES

Section 6 shows the effects of using The Auction as *bank replacement policy* in a CMP-NUCA architecture. We observed that a simple auction approach that only relocates evicted data when there is space in other banks within the NUCA cache could obtain significant performance benefits compared to prior proposals. In case that there is no space, however, the implemented approach could not determine whether the evicted data should be relocated to other bank by evicting other data block instead. For that reason, by using AUC-BASE approach almost half of started auctions finished without receiving any bid, so the evicted data could not be relocated. In this section, we propose two enhanced auction approaches – *bank usage imbalance* (AUC-ENH1-IMB) and *prioritising most accessed data* (AUC-ENH2-ACC) – that enable to determine the *quality of data* during the auction, and thus enable bidders to compare the evicted data that is being sold with their own data. By increasing the number of bids per auction, the enhanced auction approaches provide higher accuracy than AUC-BASE approach because: 1) They provide controller more options to determine the most appropriate destination bank for the evicted data within the NUCA cache, and 2) they reduce the number of auctions that finish without receiving any bid. Besides, we consider that the heuristic used by bidders in the AUC-BASE approach is basic for fairly distributing evicted data among the banks in the NUCA cache, so during an auction in both enhanced approaches, bidders will also bid for the evicted data if they have space.

7.1 Bank Usage Imbalance (AUC-ENH1-IMB)

There are two key issues when a Dynamic-NUCA (D-NUCA) architecture [22] is considered: 1) a single data can be mapped in multiple banks within the NUCA cache, and 2) the migration process moves the most accessed data to the banks that are closer to the requesting cores. Therefore, bank usage in a NUCA cache is heavily imbalanced, and a capacity miss in a heavy-used NUCA bank could cause constantly accessed data to be evicted from the NUCA cache, while other NUCA banks are storing less frequently accessed data.

We propose an auction approach that measures the usage rate of each bank. Thus, least accessed banks could bid for evicted data from banks that are being constantly accessed. We use the number of *capacity replacements* in each cache-set of NUCA banks as our bank usage metric.

This auction approach works as follows: when a replacement occurs in a NUCA bank, the owner notifies the bidders that the auction has started, and sends them the current replacement counter. Then, when a bidder receives the message from the owner, it checks whether its current replacement counter is lower than the counter attached to the message. If it is lower, the current bank bids for the evicted data by sending to the controller the bank identifier and its replacement counter. At the same time, the controller that manages the auction is storing the current winner and its replacement counter. When a bid arrives to the controller, it checks if the replacement counter from the bid is lower than the one from the current winner. If so, the incoming bid becomes the current winner, otherwise the bid is discarded. Finally, when the auction time expires, controller sends the evicted data to the bidder with the lowest replacement counter. Note that as with AUC-BASE, the auction deadline is set when the auction starts and then modified to 20 cycles when the first bid arrives.

Unfortunately, this approach is not affordable without restricting the number of bits used by each replacement counter. Therefore, in order to implement this approach, in addition to restricting the bits dedicated to the replacement counter, we have to implement a reset system that initializes the replacement counters of other NUCA banks when one of them arrives at the maximum value. If this is not done, when a replacement counter overflows, it could not be compared with other counters. Thus, when a replacement counter arrives at its maximum value, the owner sends the bidders the reset signal with the message that notifies that an auction has started.

We evaluate this approach by assuming there is an 8-bit

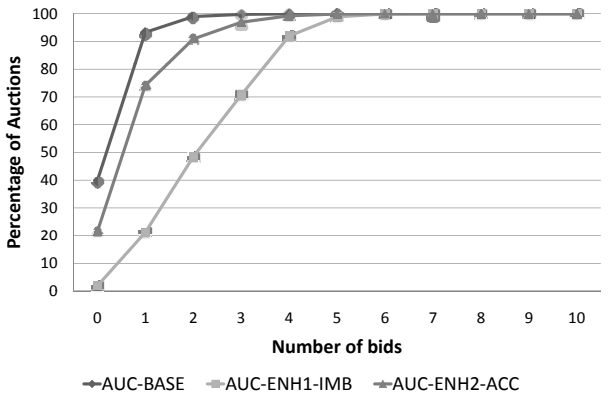


Figure 9: Received bids per auction.

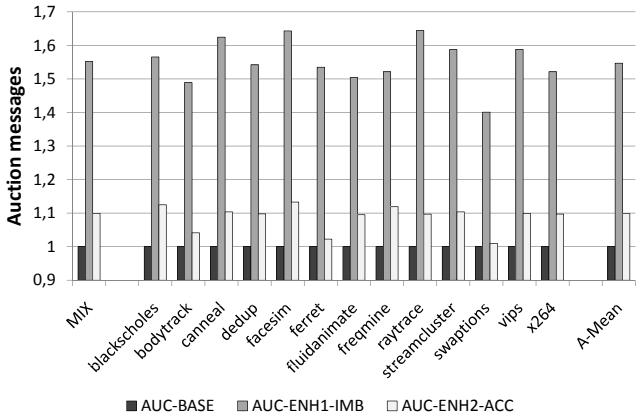


Figure 10: Auction message overhead.

replacement counter per cache-set in all NUCA banks. We have chosen this size on the basis of the following issues: additional hardware introduced (bits for the replacement counter and comparators), accuracy obtained, and reset frequency.

Hardware implementation and overheads: This approach requires the introduction of 8 bits in every cache set and auction slot. Thus, assuming the baseline architecture described in Section 2, this means adding 16.5 KBytes to the hardware overhead required by The Auction (33 KBytes). Then, this approach requires introducing 49.5 Kbytes to the 8 MByte NUCA cache, which signifies less than 0.6% of the hardware overhead. Moreover, it increases the size of both auction messages, auction invitations and bids, because these messages need to include the 8-bit replacement counter. The auction invitation message sent by the owner also requires one bit more for the reset signal. We take these overheads into account when evaluating this approach.

7.2 Prioritising most accessed data (AUC-ENH2-ACC)

This enhanced auction approach focuses on keeping the most accessed data in the NUCA cache. When the bidder receives the auction start notification, it checks whether the evicted data has been accessed more times than the data that is currently occupying the last position in the LRU-stack. If this is the case, it bids for the evicted data by sending to the controller the bank identifier and the access counter of the LRU data block. As in AUC-ENH1-IMB, when a bid arrives to the controller, it compares the access counter that comes with the incoming bid to the access counter of the current winner. If it is lower, then the incoming bid becomes the current winner. Finally, when the auction time expires, controller sends the evicted data to the bidder whose LRU data block has the lowest access counter. Note that as with AUC-BASE, the auction deadline is set when the auction starts and then modified when the first bid arrives.

This approach assumes that each line in the NUCA cache has an access counter. It only keeps information regarding accesses made to the NUCA cache, which is updated just after a hit in this cache. However, as in the previous approach, having an unbounded counter per line is not affordable, thus we assume a 3-bit saturated counter per line. We choose this size for the counter because it is sufficiently accurate, and the additional hardware introduced is still affordable.

Hardware implementation and overheads: This approach requires the introduction of 3 bits per cache line and auction slot. Thus, assuming the baseline architecture described in Section 2, it adds 49.5 KBytes to the basic auction scheme. So, the overall hardware requirements of this auction approach in the 8 MBytes NUCA cache is 82.5 KBytes, this being just 1% overhead. As in the previous proposal, the auction messages are larger. In this case, the size of the messages is increased by 3 bits. These overheads are also considered when evaluating this approach.

7.3 Analysis of results

Here, we evaluate the two enhanced auction approaches introduced in this section. Figure 9 shows that by using the AUC-BASE approach more than 40% of auctions finish without receiving any bid, and thus could not relocate the evicted data. However, we also observe that with the enhanced approaches the number of auctions that finish without bids is dramatically reduced to 0.3% and 20% by assuming AUC-ENH1-IMB and AUC-ENH2-ACC, respectively. Also note that with AUC-BASE more than 93% of auctions received just one or less bids, while assuming the same percentage with the enhanced approaches, they received up to 4. In particular, comparing both enhanced auction approaches we observe that the heuristic assumed in AUC-ENH1-IMB produces much more bids per auction than the one used in AUC-ENH2-ACC. In an auction, the more bids the controller receives the more confident its final decision will be. Increasing the number of bids per auction, however, also increases the number of messages introduced to the on-chip network. Figure 10 shows the auction message (auction invitation and bids) overhead introduced by both enhanced auction approaches. In general, we find that AUC-ENH2-ACC adds 10% more auction messages than AUC-BASE, while the overhead introduced by AUC-ENH1-IMB is 55%.

Figure 11 shows the performance results obtained with both enhanced auction approaches. In general, we observe that by increasing auction accuracy both enhanced approaches obtained significant performance benefits compared to the AUC-BASE approach in most of simulated workloads. However, we also find that network contention is a key constraint that prevents both enhanced approaches

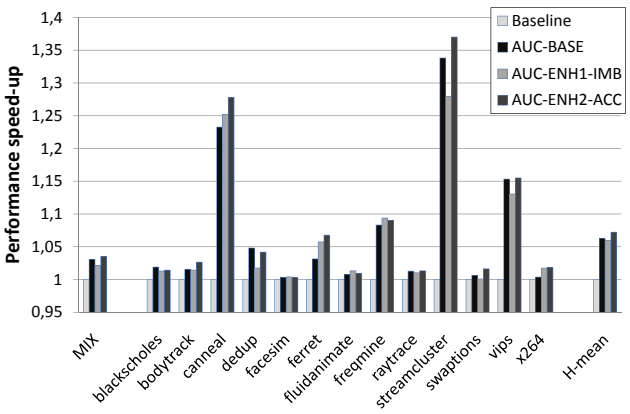


Figure 11: Performance improvement.

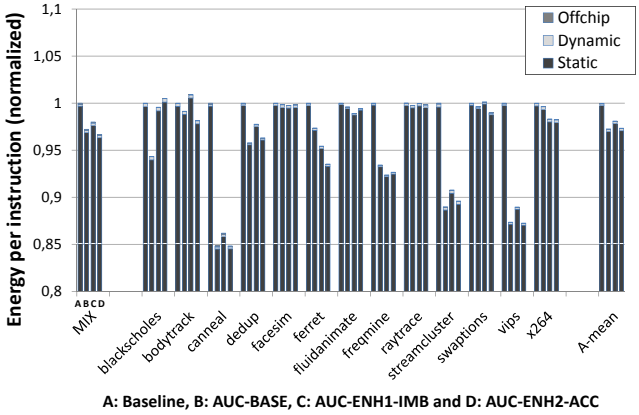


Figure 12: Energy per memory access.

achieving higher performance results. On average, AUC-ENH1-IMB and AUC-ENH2-ACC improve baseline configuration by 6% and 8%, respectively. AUC-ENH2-ACC outperforms other auction approaches in most of workloads including those with large working sets like *canneal*, *streamcluster* and *vips*. On the other hand, AUC-ENH1-IMB is heavily penalized by the high on-chip network contention that this approach introduces. Therefore, this prevents AUC-ENH1-IMB obtaining higher performance results. Actually, we experimentally observed that both enhanced auction approaches reduce NUCA miss rate obtained by AUC-BASE, but just AUC-ENH2-ACC outperformed this auction approach. Based on this observation, we conclude that the challenge to implement a high-performance auction approach is to balance *auction accuracy* and *network contention*.

Finally, Figure 12 shows the energy consumed per each memory access by both enhanced auction approaches. We observe that all auction approaches reduce energy consumed compared to the baseline configuration. With regard to the AUC-BASE, however, both enhanced approaches slightly increase energy consumed per memory access. This is because the extra on-chip network contention that both approaches introduce increases dynamic energy consumed.

8. RELATED WORK

Kim et al. [22] introduced the concept of Non-Uniform Cache Architecture (NUCA). They observed that the increase in wire delays would mean cache access times were no longer a constant. Instead, latency would become a linear-function of the line's physical location within the cache. This observation led to several NUCA architectures being designed by partitioning the cache into multiple banks and using a switched network to connect these banks. However, the two main architectures designed for this purpose were Static NUCA (S-NUCA) and Dynamic NUCA (D-NUCA). Both designs organize the multiple banks into a two-dimensional switched network. The difference between the two architectures is the *Placement Policy* they manage. While in S-NUCA architecture, data are statically placed in one of the banks and always in the same bank, in D-NUCA architecture data can be promoted to be placed in closer and faster banks. Since then, there have been several studies using NUCA architectures in the literature. One of the most relevant architectures is NuRAPID [11]. This architecture decouples data and tag placement. NuRAPID stores tags in a bank close to the processor, optimizing tag searches. Whereas NUCA searches tag and data in parallel, NuRAPID searches them sequentially. This increases overall access time but also provides greater power efficiency. Another difference between NUCA and NuRAPID is that NuRAPID partitions the cache in fewer, larger and slower banks. In terms of performance, NuRAPID and D-NUCA achieves similar results but NuRAPID heavily outperforms D-NUCA in power efficiency.

The introduction of the CMP architectures, however, brought additional challenges to the NUCA architecture, which have been analysed by Beckmann and Wood [7]. They demonstrated that block migration is less effective for CMP because 40-60% of hits in commercial workloads were satisfied in the central banks. Block migration effectively reduced wire delays in uniprocessor caches. However, to improve CMP performance, the capability of block migration relied on a smart search mechanism that was difficult to implement. Chishti et al. [11] also proposed a version of NuRAPID for CMP in which each core had a private tag array instead of a single and shared tag array.

Recent studies have explored policies for bank placement [19, 18, 17], bank migration [21, 16] and bank access [30, 4, 32] in NUCA caches. However, *The Auction* is the first mechanism proposed for NUCA caches on CMP architectures that focuses on the *bank replacement policy*. Previously, only Kim et al. [22] had proposed two alternatives for this policy, but these were based on single-processor architecture. These alternatives were *zero-copy policy* and *one-copy policy*. *Zero-copy policy* means that an evicted data element is sent back to the off-chip memory. In *one-copy policy*, on the other hand, the victim data block is moved to a lower-priority bank further from the processor.

9. CONCLUSIONS AND FUTURE WORK

The decentralized nature of NUCA prevents the replacement policies proposed in the literature from being effective in this kind of caches. As banks operate independently from each other, their replacement decisions are restricted to a single NUCA bank. Unfortunately, *bank replacement*

policy in NUCA-based CMP machines has not been properly addressed before. Most previous works have ignored the replacement issue or adopted a replacement scheme that was originally designed for use with uniprocessors/uniform-caches. To the best of our knowledge, *The Auction* is the first mechanism that specifically targets CMP-NUCA systems, thus accounting for the decentralized nature of the cache and the specific characteristics that result from the multi-threaded, multi-banked cache system.

The Auction *spreads* replacement decisions that have been taken in a single bank to the whole NUCA cache. This enables the most appropriate data to be evicted from the NUCA cache. In this paper we show that The Auction is a sophisticated, adaptive and flexible mechanism that enables evicted lines to relocate from a bank in the NUCA cache to the most suitable bank. It defines a protocol that is based on decisions taken at three different points: *the owner*, *the controller* and *the bidders*. The owner, which is the bank where the initial replacement occurs, starts the auction by notifying the replacement to all bidders. The bidders, on the other hand, are the banks where the evicted data can be mapped into the NUCA. They decide whether are interested in obtaining the current evicted data. Finally, the controller decides which bank is going to be the final destination of the evicted data.

The flexibility of the auction lies in the decisions that its components take. In this paper, we analyse three different approaches: *AUC-BASE*, *AUC-ENH1-IMB* and *AUC-ENH2-ACC*. By using the first one, we showed the benefits in terms of performance and energy consumption that a simple auction approach could provide to a CMP-NUCA architecture. Then, we proposed two enhanced auction approaches to increase the number of bids per auction, and thus make final replacement decision much more accurate. At this point, we observe that although increasing auction accuracy could improve performance (by 8%, on average), network contention is a key constraint that must be considered. Furthermore, we also found that the auction reduces the energy consumed per memory access by 4%.

Future research will include analysing other auction approaches to optimize the trade-off between auction accuracy and network contention. Increasing the accuracy of the owner decisions will reduce the number of messages introduced by the auction to the network. Therefore, energy consumption should be reduced and could even improve performance obtained by the current implemented auction approaches. On the other hand, smarter decisions by the controller component will enable evicted data to be sent to the most suitable bidders. Consequently, they may significantly increase the current performance improvement achieved. Furthermore, further research should also involve analysing the impact of adopting adaptive bidder decisions that can adapt to the current program behaviour.

10. ACKNOWLEDGEMENTS

This work is supported by the Spanish Ministry of Science and Innovation (MCI) and FEDER funds of the EU under the contracts TIN 2007-61763 and TIN 2007-68050-C03-03, the Generalitat de Catalunya under grant 2009SGR1250, and Intel Corporation. Javier Lira is supported by the MCI under FPI grant BES-2008-003177.

11. REFERENCES

[1] Micron system power calculator. [Online]. Available: $http: //www.micron.com/support/part_info/powercalc$

[2] Spec cpu2006. [Online]. Available: http://www.spec.org/cpu2006

[3] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate vs. ipc: The end of the road for conventional microprocessors," in *Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[4] S. Akioka, F. Li, K. Malkowski, P. Raghavan, M. Kandemir, and M. J. Irwin, "Ring data location prediction scheme for non-uniform cache architectures," in *Proocedings of the International Conference on Computer Design*, 2008.

[5] A. Bardine, P. Foglia, G. Gabrielli, and C. A. Prete, "Analysis of static and dynamic energy consumption in nuca caches: Initial results," in *Proceedings of the 2007 Workshop on Memory Performance: Dealing with Applications, Systems and Architecture*, 2007.

[6] B. M. Beckmann, M. R. Marty, and D. A. Wood, "Asr: Adaptive selective replication for cmp caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium of Microarchitecture*, 2006.

[7] B. M. Beckmann and D. A. Wood, "Managing wire delay in large chip-multiprocessor caches," in *Proceedings of the 37th International Symposium on Microarchitecture*, 2004.

[8] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[9] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in *Proceedings of the 33rd International Symposium on Computer Architecture*, 2006.

[10] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," in *Proceedings of the 21st ACM International Conference on Supercomputing*, 2007.

[11] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance associativity for high-performance energy-efficient non-uniform cache architectures," in *Proceedings of the 36th International Symposium on Microarchitecture*, 2003.

[12] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Optimizing replication, communication, and capacity allocation in cmps," in *Proceedings of the 32nd International Symposium on Computer Architecture*, 2005.

[13] P. Dubey, "A platform 2015 workload model: Recognition, mining and synthesis moves computers to the era of tera," in *Intel White Paper*, Intel Corporation, 2005.

[14] H. Dybdahl and P. Stenström, "An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors," in *Proceedings of the 13th International Symposium on High-Performance Computer Architecture*, 2007.

[15] E. Grochowski, R. Ronen, J. Shen, and H. Wang,

"Best of both latency and throughput," in *Proceedings of the 22nd International Conference on Computer Design*, 2004.

[16] M. Hammoud, S. Cho, and R. Melhem, "Acm: An efficient approach for managing shared caches in chip multiprocessors," in *Proceedings of the 4th International Conference on High Performance and Embedded Architectures and Compilers (HiPEAC)*, 2009.

[17] M. Hammoud, S. Cho, and R. Melhem, "Dynamic cache clustering for chip multiprocessors," in *Proceedings of the ACM International Conference on Supercomputing (ICS)*, 2009.

[18] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive nuca: Near-optimal block placement and replication in distributed caches," in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*, 2009.

[19] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A nuca substrate for flexible cmp cache sharing," in *Proceedings of the 19th ACM International Conference on Supercomputing*, 2005.

[20] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," in *Proceedings of the 17th annual international symposium on Computer Architecture*, ISCA '90.

[21] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son, "A novel migration-based nuca design for chip multiprocessors," in *Proceedings of the ACM/IEEE conference on Supercomputing*, 2008.

[22] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.

[23] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: a 32-way multithreaded sparc processor," in *IEEE Micro*, March 2005.

[24] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "Ibm power6 microarchitecture," *IBM Journal*, November 2007.

[25] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, *Simics: A Full System Simulator Platform.* Computer, 2002, vol. 35-2, pp. 50–58.

[26] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in *Proceedings of the 30th International Symposium on Computer Architecture*, 2003.

[27] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," in *Computer Architecture News*, Sep. 2005.

[28] D. Matzke, "Will physical scalability sabotage performance gains?" *IEEE Computer*, September 1997.

[29] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-thread itanium processor," *IEEE Micro*, vol. 25, no. 2, March-April 2005.

[30] N. Muralimanohar and R. Balasubramonian, "Interconnect design considerations for large nuca caches," in *Proceedings of the 34th International Symposium on Computer Architecture*, 2007.

[31] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007.

[32] R. Ricci, S. Barrus, and R. Balasubramonian, "Leveraging bloom filters for smart search within nuca caches," in *Proceedings of the 7th Workshop on Complexity-Effective Design (WCED)*, 2006.

[33] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finnan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28tflops network-on-chip in 65nm cmos," in *Proceedings of the IEEE International Solid-State Circuits Conference*, 2007.

[34] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "Simflex: Statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[35] M. Zhang and K. Asanović, "Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors," in *Proceedings of the 32nd International Symposium on Computer Architecture*, 2005.