HK-NUCA: Boosting Data Searches in Dynamic Non-Uniform Cache Architectures for Chip Multiprocessors

Javier Lira
Dept. of Computer Architecture
Universitat Politècnica de Catalunya
08034 Barcelona, Spain
javier.lira@ac.upc.edu

Carlos Molina
Dept. of Computer Engineerging
Universitat Rovira i Virgili
43007 Tarragona, Spain
carlos.molina@urv.net

Antonio González
Intel Barcelona Research Center
Intel Labs - UPC
08034 Barcelona, Spain
antonio.gonzalez@intel.com

Abstract*—The exponential increase in the cache sizes of multicore processors (CMPs) accompanied by growing on-chip wire delays make it difficult to implement traditional caches with single and uniform access latencies. Non-Uniform Cache Architecture (NUCA) designs have been proposed to address this problem. 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. Traditionally, NUCA organizations have been classified as static (S-NUCA) and dynamic (D-NUCA). While in S-NUCA a data block is mapped to a unique bank in the NUCA cache, D-NUCA allows a data block to be mapped in multiple banks. Besides, D-NUCA designs are dynamic in the sense that data blocks may migrate towards the cores that access them most frequently. Recent works consider D-NUCA as a promising design, however, in order to obtain significant performance benefits, they used a non-affordable access scheme mechanism to find data in the NUCA cache. In this paper, we propose a novel and implementable data search algorithm for D-NUCA designs in CMP architectures, which is called HK-NUCA (Home Knows where to find data within the NUCA cache). It exploits migration features by providing fast and power efficient accesses to data which is located close to the requesting core. Moreover, HK-NUCA implements an efficient and cost-effective search mechanism to reduce miss latency and on-chip network contention. We show that using HK-NUCA as data search mechanism in a D-NUCA design reduces about 40% energy consumed per each memory request, and achieves an average performance improvement of 6%.

***Keywords*-D-NUCA; access; cache memory; CMP;**

I. Introduction

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. However, the exponential increase in the cache sizes on multicore processors accompanied by growing on-chip wire delays [1] make it difficult to implement traditional caches with single and uniform access latencies. Non-Uniform Cache Architecture (NUCA) designs [2] have been proposed 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.

Traditionally, NUCA designs have been classified as static (S-NUCA) and dynamic (D-NUCA). While in S-NUCA a data block is mapped to a unique bank in the NUCA cache, D-NUCA allows data to be mapped in multiple banks. By enabling data to reside in multiple banks within the NUCA cache, D-NUCA attempts to improve performance by leveraging locality and moving most accessed data to banks that are close to the processors. Previous works agree that D-NUCA is a promising design, however, they also show that D-NUCA could not obtain significant performance benefits unless a non-affordable access scheme is utilized to find data in the NUCA cache [3], [4], [2].

Dynamic features provided by D-NUCA, like multiple placement of data and migration movements, make data search scheme a key constraint in this kind of architectures. Prior works proposed different techniques to efficiently search data in the NUCA banks which include access parallelization, prioritisation of most frequently accessed banks, or prediction of the next bank to access, among others [2], [5], [6]. In case of miss in the NUCA cache, however, these techniques must access all banks from the corresponding bankset to ensure that the requested data block is not in the NUCA cache before forwarding the request to the upper-level memory. In contrast, HK-NUCA, which is short for (*Home Knows where to find data within the NUCA cache*), uses *home knowledge* to ascertain which banks could potentially have the requested data, thus it resolves memory requests by accessing only to the subset of banks indicated by home. This makes HK-NUCA significantly reduce on-chip network contention as well as miss resolution time compared to other techniques previously proposed in the literature. This translates into a reduction of the dynamic energy consumed by the memory system of 40%, and an overall performance improvement of 6%.

The remainder of this paper is structured as follows. Section II describes the baseline architecture assumed in

this paper. Section III lays out the motivation for this work by analysing the importance of data search algorithm in D-NUCA. In Section IV, the proposed data search mechanism is described in detail. Section V presents the experimental method we used, followed by the analysis of the results that is presented in Section VI. Related work is discussed in Section VII, and concluding remarks are given in Section VIII.

II. Baseline Architecture

We assume a Non-Uniform Cache Architecture (NUCA) L2 cache, derived from Kim et al.'s Dynamic NUCA (D-NUCA) design [2]. Similar to the original proposal we partition the address space across cache banks, which are connected via a 2D mesh interconnection network. As illustrated in Figure 1, the NUCA storage is partitioned into 128 banks. D-NUCA allows for migration movements which distribute data blocks among the NUCA banks in order to have the most accessed data close to the cores, and thus reduce access latency for future accesses to the same data. Ideally, a data block could be mapped into any cache bank in order to maximize placement flexibility, however, the overhead of locating a data block may be too large as each bank must be searched, either through a centralized tag store or by broadcasting the tags to all the banks. To address this situation all banks that compose the NUCA cache are treated as a set-associative structure, and each bank holds one "way" of the set, which are called *banksets*. Thus, data blocks can only be mapped to one bankset. The NUCA banks that compose a bankset are organized among the NUCA cache in bankclusters (dotted boxes in Figure 1). Each bankcluster consists of a single bank of each bankset. As shown in Figure 1, we assume a NUCA cache 16-way bankset associative that is organized in 16 bankclusters, eight are located close to the cores (local banks) and the other eight in the center of the NUCA cache (central banks). Therefore, a data block has *16 possible placements in the NUCA cache* (eight *local banks* and eight *central banks*).

The bank where an incoming data block is going to be mapped when it comes from the off-chip memory is statically predetermined based on the lower bits of the data block's address. This is its *home* bank. The LRU data block in this bank would be evicted if it is needed. Once the data block is in the NUCA cache, the migration scheme will determine its optimal position in there. As migration policy, we assume *gradual promotion* that has been widely used in the literature [2], [3]. It defines that upon a hit in the NUCA cache, the requested data block should move one-step closer to the core that initiate the memory request. With regard to the *data search* scheme, the baseline D-NUCA design uses a two-phase multicast algorithm, that is also known as *partitioned multicast*. First, it broadcasts a request to the *local bank* that is closest to the processor that launched the memory request, and to the eight *central banks*. If all nine

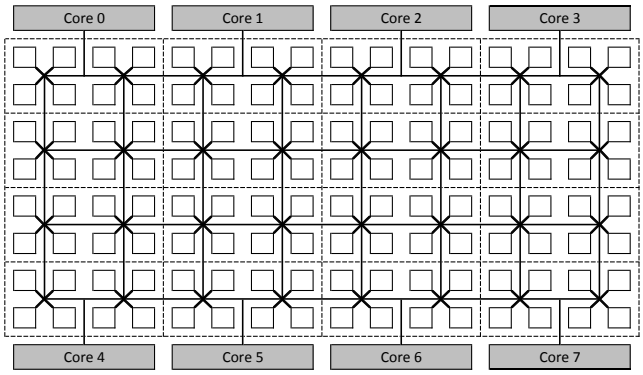


Figure 1. Baseline architecture layout.

initial requests miss, the request is sent, also in parallel, to the remaining seven banks from the requested data's bankset. Finally, if the request misses all 16 banks, the request would be forwarded to the off-chip memory.

HK-NUCA is a data search mechanism for D-NUCA designs. Therefore, when we evaluate HK-NUCA further in the paper, we will assume a NUCA architecture as the one described in this section, but using the HK-NUCA algorithm to find data in the cache instead of the partitioned multicast.

III. Background and Motivation

NUCA designs have been classified on the basis of their placement policy as static (S-NUCA) or dynamic (D-NUCA) [3], [2]. In an S-NUCA organization, the mapping of data into banks is predetermined based on the block index, and thus can reside in only one bank of the cache. This facilitates the data search algorithm, since the cache controller sends the request to a single bank. On the other hand, because of the static placement within the NUCA cache, access latency to a specific data block is essentially decided by its address. So, if frequently accessed data blocks are mapped to banks with longer latencies, S-NUCA will not provide optimal performance.

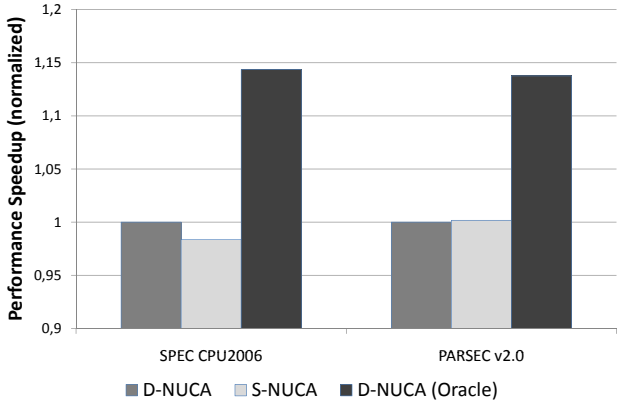
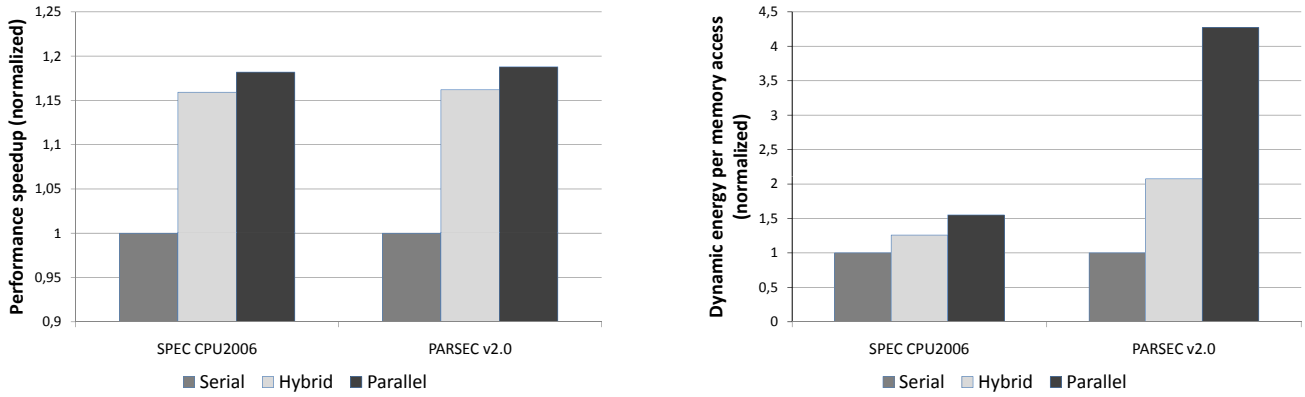


Figure 2. Performance results of both NUCA organizations, S-NUCA and D-NUCA.



(a) Performance speedup

(b) Energy consumed by the memory system

Figure 3. Impact of several access schemes in a D-NUCA architecture.

With regard to D-NUCA, it allows data blocks to be mapped to multiple candidate banks within the NUCA cache. With proper placement and migration policies, D-NUCA attempts to improve performance by leveraging locality and moving recently accessed data blocks to NUCA banks near the processors that are requesting them, thus reducing cache access latency with frequently accessed data. However, enabling multiple destinations for a single data block within the NUCA cache creates two new challenges. First, many banks may need to be searched to find a data block on a cache hit. Second, if the data block is not in the cache, the slowest bank determines the time necessary to ascertain that the request is a miss. So, bank access policy is a key constraint in this NUCA architecture. In fact, recent works have considered D-NUCA a promising mechanism [3], [4], [2] but, they have also shown that D-NUCA cannot obtain significant performance benefits unless an unaffordable access scheme is used to locate data in the NUCA cache.

Figure 2 shows the performance results obtained with both NUCA architectures[1], S-NUCA and D-NUCA. We found that D-NUCA and S-NUCA achieve similar performance results in both multi-threaded and multi-programmed environments. We also evaluated D-NUCA but using an oracle as access scheme. It outperformed D-NUCA with a partitioned-multicast search algorithm by almost 15%. These results prove that although D-NUCA can reduce cache access latencies by exploiting locality, this organization is heavily penalized by the data search algorithm employed.

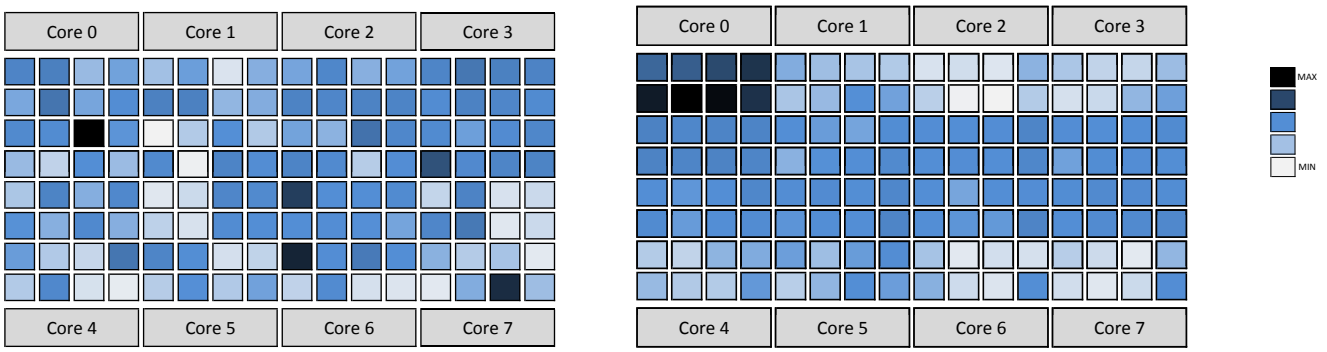
[1]The experimental methodology followed in this section is described in Section V.

A. Locating data within the NUCA cache

Traditionally, there are two distinct access policies to search for a data block within the NUCA cache: *serial* and *parallel* access [2]. Assuming *serial access*, the NUCA banks that compose the corresponding *bankset* are sequentially accessed – starting from the closest bank to the requesting core – until the requested data block is found or a miss occurs in the last NUCA bank. This policy minimizes the number of messages in the cache network and keeps energy consumption low, at the cost of reduced performance. On the other hand, using *parallel access*, all the NUCA banks in the *bankset* are accessed in parallel, thereby offering higher performance at the cost of increased energy consumption and network contention.

Hybrid access policies that try to have the best of both worlds – high performance and low energy consumption – have also been explored [3], [2]. The best performing hybrid access scheme is *partitioned multicast*, which is used in the baseline architecture II. The differences in terms of performance and energy consumption of using one of the access schemes are shown in Figure 3(a) and Figure 3(b), respectively. As expected, the serial approach reduces the dynamic energy consumed by the memory system; however, it also results in significant performance degradation compared to the other two access policies. On the other hand, although parallel access clearly outperforms the other two evaluated access approaches, we have found that the high network contention caused by accessing all NUCA banks in parallel, prevents parallel access achieving higher performance benefits.

Prior works also proposed introducing a partial tag structure to reduce the miss resolution time [4], [2]. Before accessing the NUCA cache, the stored partial tag bits are compared with the corresponding bits of the requested tag,



(a) Static NUCA (S-NUCA)

(b) Dynamic NUCA (D-NUCA)

Figure 4. Distribution of hits in the NUCA cache when the core 0 sends memory requests.

and if no matches occur, the miss processing is commenced early. Otherwise, the NUCA cache is accessed following the implemented data search algorithm. Kim et al. [2] showed that this mechanism effectively accelerated NUCA accesses to single-processor D-NUCA caches. When a CMP architecture is considered, however, the implementation of this mechanism appears impractical [3]. Nevertheless, Huh et al. [4] did it, and showed that the benefits achieved were not worth its complexity and hardware requirements.

Recent works have proposed *predicting* data location within the NUCA cache to obtain the high performance benefits of an *oracle* at a lower cost [5], [6]. Unfortunately, predicting data location within the NUCA cache has proved to be an extremely difficult task, so much so that in most cases the benefits achieved are not worth the significant cost and complexity in hardware that such mechanisms require.

Note that in case of miss in the NUCA cache, all techniques introduced in this section must access all banks from the corresponding bankset to ensure that the requested data block is not in the NUCA cache before forwarding the request to the upper-level memory. In contrast, HK-NUCA uses *home knowledge* to ascertain which banks could potentially have the requested data, thus it resolves memory requests by accessing only to the subset of banks indicated by home.

B. Exploiting spatial/temporal locality

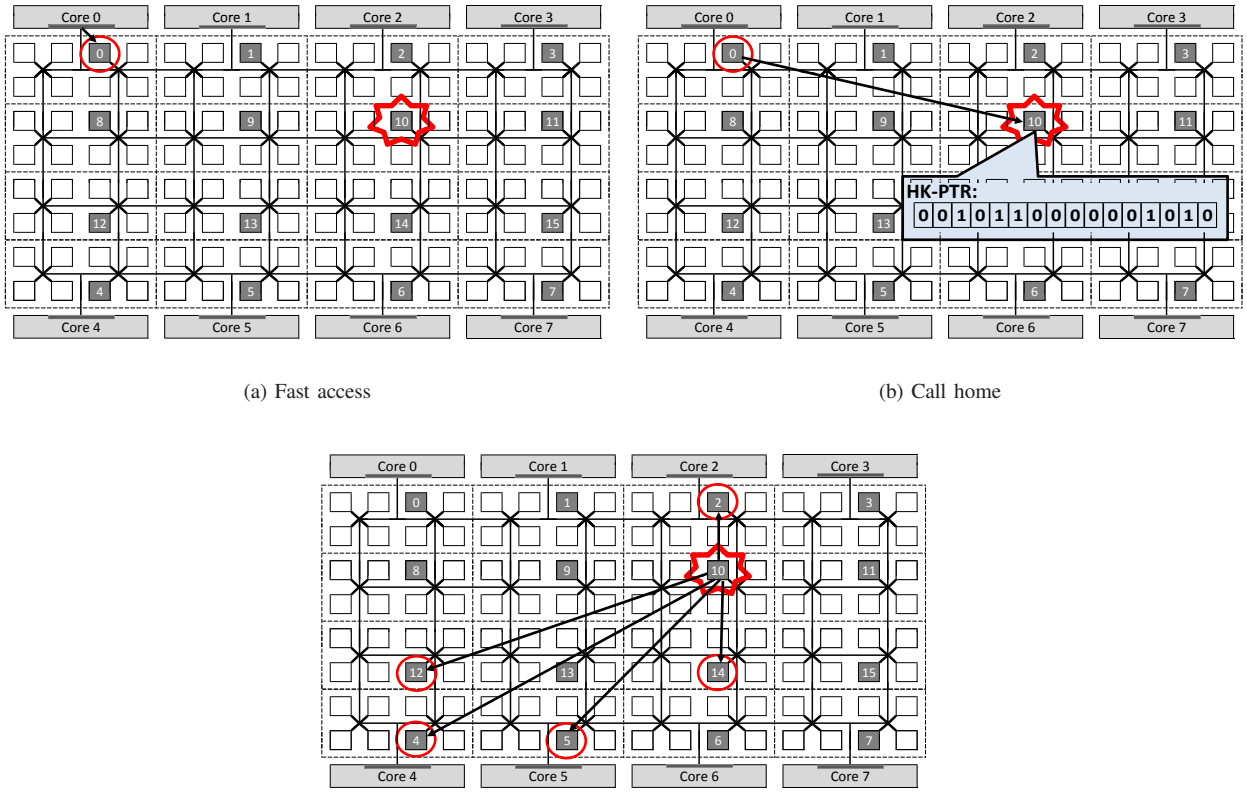
NUCA designs that allow data blocks to be mapped in multiple banks (e.g. D-NUCA) take advantage of spatial/temporal locality in memory accesses to move most frequently accessed data closer to the requesting core, thus optimizing cache access latency for future accesses. Figure 4 clearly shows the effects of the migration movements by illustrating how hits on the NUCA cache are distributed among the NUCA banks. On one hand, S-NUCA provides a fair hit distribution among all NUCA banks. Thanks to migration movements, on the other hand, when a D-NUCA architecture is assumed most hits in the NUCA cache are served by the banks closer to the processors. Furthermore, Figure 4 also indicates that migration movements effectively help D-NUCA to exploit the spatial/temporal locality found in most applications, meaning the data search algorithm is the primary drawback in these type of architectures.

In this paper, we propose a novel *bank access policy* for CMP-NUCA architectures called *HK-NUCA*. It exploits migration features by providing fast and energy-efficient access to data which is located close to the requesting core. Moreover, *home knowledge* facilitates data searches by giving hints to the access algorithm to resolve memory accesses in a fast and efficient way.

IV. HK-NUCA: Home Knows where to find data within the NUCA cache

This section describes the novel search mechanism for efficiently locating data in CMP-NUCA architectures that we propose in this paper. This mechanism is called *HK-NUCA*, which is short for *Home Knows where to find data within the NUCA cache*. Before describing the mechanism, the term *home* should first be introduced to better understand the remainder of this section. A data block can reside in any of the banks that compose a bankset, but only one of them is its *home* NUCA bank. Data-home relationship is statically predetermined based on the lower bits of the data block's address. Consequently, all banks in the NUCA cache act as home, and each of them manages the same number of data blocks.

The principal function of *home* is to know which other NUCA banks have at least one of the data blocks that it



(a) Fast access

(b) Call home

(c) Parallel access

Figure 5. Scheme of HK-NUCA data search mechanism.

manages. In order to keep this information, all NUCA banks have a set of *HK-NUCA pointers (HK-PTRs)*. HK-PTR assigns a single bit to each bank from the bankset – 16 bits in our case –. If a bit in the HK-PTR is set (1), this means that the corresponding NUCA bank is currently storing at least one of the data blocks the current bank manages. In contrast, if it is not set (0), the corresponding NUCA bank does not have any data blocks from the current home bank. There is a HK-PTR for each cache-set in the NUCA bank. We find that pointing at the cache-set level significantly increases the global accuracy of HK-NUCA by reducing the total amount of 1's per HK-PTR.

This mechanism, based on the knowledge of *home banks*, significantly reduces on-chip network traffic as well as the miss resolution time, and consequently provides faster and more efficient accesses to the NUCA cache. The following section describes how HK-NUCA search mechanism works, Section IV-B details HK-PTR management tasks, and Section IV-C introduces the implementation details.

A. How HK-NUCA works

After getting a miss in a private first-level cache from any core, the NUCA cache is probed by means of the *bank access policy* that we propose in this paper. HK-NUCA consists of the following three stages: *1) Fast access, 2) Call home, and 3) Parallel access*.

Fast access: This is the first stage of HK-NUCA (Figure 5(a)). As described in Section III-B, migration moves most accessed data to banks that are closer to the requesting cores. In order to take advantage of this feature, the cache controller of the requesting core first accesses the closest NUCA bank where the requested data block can be mapped within the NUCA cache. In case of a hit, the *fast access* stage will respond to the memory request by taking minimum access latency and introducing just one message into the on-chip network. If the requested data is not found, however, the memory request is forwarded to the requested data's home bank by starting the next HK-NUCA stage.

Call home: This is the second stage of HK-NUCA. As Figure 5(b) shows, after the *fast access* stage, the requested data's home NUCA bank is accessed. If the requested data is found, it is sent to the core that started the memory request, and the search is completed. If a miss occurs, however, the home bank retrieves the corresponding *HK-PTR* to find out which NUCA banks may have the requested data. Then the memory request is sent, in parallel, to all candidate

NUCA banks which have their bit set in the HK-PTR (Figure 5(c)). However, note that thanks to the *home* knowledge, the memory request will be sent in parallel to only a few banks, rather than to all of the 14 non-visited banks.

Parallel access: This is the final stage of HK-NUCA. If the requested data is still not found, the memory request is finally sent to the off-chip memory. Otherwise, the NUCA bank in which the hit occurs sends the requested data to the core that started the memory request.

Furthermore, there are some particular cases that HK-NUCA considers that could accelerate the data search. For instance, when the bank accessed in the *fast access* stage actually is the requested data block's home bank, the two first HK-NUCA stages are combined. Particular cases also include removing parallel access if none of the bits from HK-PTR are set in the *call home* stage. In this case, the memory request would be directly sent to the main memory instead. Finally, in order to avoid redundant accesses, during the *parallel access* stage HK-NUCA will not send memory requests to the bank which has been previously accessed during the *fast access* stage, even if its corresponding bit in HK-PTR is set. In Section VI, we also evaluate a variation of the HK-NUCA algorithm that launches the *fast access* and *call home* stages in parallel. This accelerates miss resolution time of the HK-NUCA, at the cost of increasing on-chip network contention.

B. Managing Home Knowledge

Keeping updated HK-PTRs is crucial for ensuring the efficiency and accuracy of the HK-NUCA data search mechanism. When a NUCA bank allocates a data block, a notification message is sent to its home bank in order to set the corresponding bit in the HK-PTR. On the other hand, if it is evicted, the current bank notifies the evicted data block's home bank to reset the specific bit in the HK-PTR. In order to reduce on-chip network traffic due to coherence in HK-NUCA, before notifying the home banks, the current bank checks whether it is already storing a data block from the incoming/evicted data block's home bank. If so, it is not necessary to notify the home bank since its corresponding bit of HK-PTR is already updated. Otherwise, notification is sent.

There are three actions that may provoke an update in HK-PTR: 1) a new data enters to the cache, 2) an eviction from a NUCA bank, and 3) a migration movement. As described in Section II, when an incoming data enters to the NUCA cache from the upper-level memory, it is mapped to its *home* NUCA bank, thus updating HK-PTR is not necessary. If there is an eviction in a NUCA bank and HK-PTR must be updated, a notification message is sent to the corresponding home bank. The corresponding HK-PTR, however, would not be updated until the notification message arrives to the home bank. Then, if the non-updated HK-PTR is accessed to start the *parallel access* stage at this point,

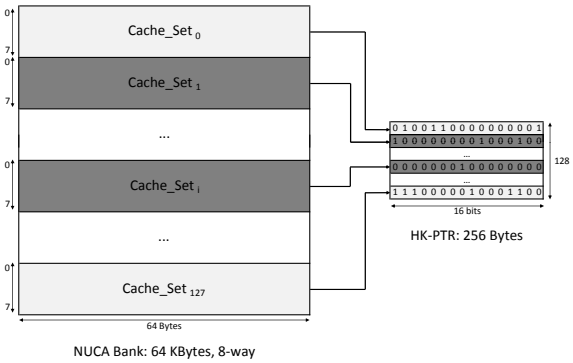


Figure 6. HK-NUCA pointer list (HK-PTR).

HK-NUCA would send an unnecessary memory request. Although this misalignment may provoke slightly increasing on-chip network traffic, it does not affect to the correctness of the HK-NUCA algorithm. Migration movements may provoke up to two HK-PTR updates, one for each data block involved in the migration process. Contrary to the eviction action, HK-PTR updates sent during the migration process may indicate that a new bank has a data block that the home actually manages. In this case, note that if the non-updated HK-PTR is used, before receiving the notification message, to find the same data block that is being migrated, HK-NUCA will not be able to find the requested data. In order to avoid this situation and ensure correctness of the HK-NUCA algorithm, we synchronize migration movements with HK-PTR updates.

C. Implementing HK-NUCA

HK-NUCA requires introducing some additional hardware to implement HK-PTRs. Figure 6 shows that HK-PTR needs 256 bytes per NUCA bank, so assuming that our 8 MByte-NUCA cache consists of 128 banks, the total hardware required by HK-PTRs is 32 KBytes which is less than 0.4% of hardware overhead. Apart from hardware overhead due to HK-PTRs, HK-NUCA also introduces some complexity in the NUCA design (e.g. comparators), but their timing effects are hidden by the critical path. These implementation overheads (hardware and design complexity), however, are accurately modeled in our experimental evaluation, therefore the results shown in Section VI already take these issues into consideration.

Although HK-NUCA is proposed in a heavy-banked NUCA cache, this is not the only organization it can work with. HK-NUCA, for instance, can be easily adapted to a 4x4 tiled-CMP architecture – similar to that assumed in some recent works in the literature [7], [8], [9]–. As with the heavy-banked architecture, the 4x4 tiled-CMP allows a data block to be mapped in 16 positions within the NUCA cache. In this case, HK-PTRs are as big as with our baseline, so the extra hardware required would be the same. Moreover, we believe that the benefits in terms of performance and

energy consumption obtained using HK-NUCA on the tiled architecture would be similar to that achieved with our baseline because both architectures are D-NUCA designs.

V. Experimental Methodology

A. Simulation Environment

We use the full-system execution-driven simulator, Simics [10], extended with the GEMS toolset [11]. 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. In order to maintain coherency along the memory subsystem, we used the MESIF coherence protocol, which is also used in the Intel® Nehalem processor [12]. Table I 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 [13] 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, 128 Banks
NUCA Bank	64 KBytes, 8-way
L1 Latency	3 cycles
NUCA Bank Latency	4 cycles
Router Latency	1 cycle
Avg Offchip Latency	250 cycles

Table I
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 [14] workloads with the *reference* input. Table II outlines the workloads that make up this scenario. The latter simulates the whole set of applications from the PARSEC v2.0 benchmark suite [15] 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.

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 [16].

astar	gcc	lbm	mcf
milc	omnetpp	perlbench	soplex
Reference input			

Table II
Multi-programmed scenario.

B. 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. [17]. This allowed us to also consider the 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:

$$E_{dynamic} = E_{network} + E_{banks} + E_{off-chip}$$

The total dynamic energy consumed is the sum of the dynamic energy consumed by the NUCA cache within the chip ($E_{network}$ and E_{banks}) and the energy dissipated per each access to the off-chip memory ($E_{off-chip}$). In order to compute E_{banks}, CACTI 6.0 [13] has been used to evaluate dynamic energy consumed per each memory access, while GEMS [11] provided the dynamic behaviour in the applications. GEMS also integrates a power model based on Orion [18] that we used to evaluate the dynamic power consumed by the on-chip network ($E_{network}$). Note that the extra messages introduced by HK-NUCA into the on-chip network has also been accurately modeled into the simulator. The energy dissipated by the off-chip memory ($E_{off-chip}$) was determined using the *Micron System Power Calculator* [19] assuming a modern DDR3 system (4GB, 8DQs, Vdd:1.5v, 333 MHz). Our evaluation of the off-chip memory focused on the energy dissipated during active cycles and isolated 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 memory access. It is based on the energy per instruction (EPI) [20] metric which is commonly used for analysing the energy consumed by the whole processor. This metric works independently of the amount of time required to process an instruction and is ideal for throughput performance.

VI. Results and Analysis

The primary characteristics of HK-NUCA make it a promising data search mechanism that not only boosts performance, but also reduces energy consumption. On one

hand, by taking advantage of migration movements, the *fast access* stage of HK-NUCA provides high performance at a minimum cost. *Home knowledge*, on the other hand, allows memory requests to be satisfied by only accessing the NUCA banks that can potentially have the requested data. With that, HK-NUCA significantly reduces on-chip network traffic and diminishes router delays due to congestion. This section analyses the impact on performance and energy consumption of using HK-NUCA as a *bank access policy* in a dynamic CMP-NUCA architecture. We evaluate two versions of the HK-NUCA algorithm. One follows the three-step data search algorithm described in Section IV. The other launches the *fast access* and the *call home* stages in parallel, turning HK-NUCA into a 2-step algorithm. In the remainder of the section, we call them HK-NUCA (3-steps) and HK-NUCA (2-steps), respectively. We compare HK-NUCA with another two bank access policies for D-NUCA architectures: 1) partitioned multicast, and 2) perfect search. The former searches for data in the NUCA cache in two steps. It first accesses in parallel a subset of NUCA banks, which includes the closest bank to the requesting core. If the requested data is not found the other NUCA banks from the bankset are then accessed in parallel. Recent works have considered partitioned multicast as the best performing access policy, achieving significant performance benefits without incurring unaffordable energy consumption [3], [2]. The other access policy we evaluated is perfect search, which works as an oracle. It always knows whether the requested data block is in the NUCA cache. If so, perfect search directly sends the memory request to the corresponding NUCA bank. However, this is an unrealistic access scheme. Comparing HK-NUCA with this mechanism will show how far it is from the optimal approach.

A. Performance analysis

Figure 7 shows the performance improvement achieved with the four configurations that we evaluated: partitioned multicast, HK-NUCA (3-steps), HK-NUCA (2-steps) and perfect search. On average, we found that both versions of HK-NUCA, 3-steps and 2-steps, outperform partitioned multicast access scheme by 4% and 6%, respectively. In general, the HK-NUCA access mechanism performs significantly well with most PARSEC applications, by achieving about 10% performance improvement in five of them (*canneal*, *ferret*, *freqmine*, *raytrace* and *streamcluster*). On the other hand, assuming the multi-programmed environment (SPEC CPU2006 in Figure 7), HK-NUCA was found to improve performance by an average of 10%.

Perfect search shows the optimal performance that can be achieved in a D-NUCA organization by only modifying the *bank access policy*. As an oracle, it knows whether the requested data block is in the NUCA cache. If so, the memory request is directly sent to the corresponding NUCA bank. Otherwise, the memory request is forwarded to the upper-level memory. Figure 7 shows that perfect search outperforms partitioned multicast access scheme by 14%. In general, HK-NUCA (2-steps) obtains about half performance benefits than perfect search does. Moreover, HK-NUCA (2-steps) achieves similar performance results to perfect search assuming workloads with small working sets, such as *blackscholes* and *x264*.

In HK-NUCA, memory requests are resolved during one of the three stages that compose this mechanism: *fast access*, *call home* and *parallel access*. Note that while satisfying the memory request during the two first stages signify getting a hit on the NUCA cache, memory requests resolved during the *parallel access* stage may either be hits on this stage, or misses. Figure 8 illustrates the percentage of memory requests satisfied by each HK-NUCA stage. Moreover, the hit rate in the NUCA cache for each benchmark is indicated by the height of the corresponding bar. Note that assuming the 2-step version of the HK-NUCA algorithm, Figure 8 would be the same, but combining the *fast access* and *call home* stages.

We define three different scenarios to analyse the performance results achieved with HK-NUCA. Figure 7 and Figure 8 shows that HK-NUCA significantly outperforms (up to 18%) partitioned multicast with those workloads that have high miss rate in the NUCA cache, like *canneal*, *streamcluster*, *vips* or the multi-programmed workload *SPEC CPU2006*. Because of *home knowledge* provided by the HK-NUCA algorithm, it does not have to access all banks from the bankset before accessing to the upper-level memory, thus the miss resolution time is reduced significantly. The higher the miss rate in the NUCA cache, the bigger the impact this situation has on performance. The second scenario we found includes those workloads with low miss rate in the NUCA cache, but high rate (50% or more) of hits resolved during the first two stages (*fast access* and *call home*): e.g. *dedup*, *raytrace*, *swaptions*, or *x264*. Both partitioned multicast and HK-NUCA take similar access latencies to satisfy memory requests when they hit on the closest NUCA bank. While the former accesses this bank during its first step, HK-NUCA accesses it during the *fast access* stage. Thus, partitioned multicast introduces nine messages to the on-chip network, whereas HK-NUCA needs just one (or two with the 2-steps version). Therefore, HK-NUCA reduces on-chip network traffic and diminishes router delays due to congestion which result to be key factors to outperform (by 5%) partitioned multicast in this scenario. The third scenario is composed of *bodytrack* and *facesim*. In this case, HK-NUCA achieves 4% performance loss compared to the partitioned multicast scheme. The principal characteristics of this scenario are, on one hand, very high hit rate in the NUCA cache (close to 99%) and, on the other hand, most of memory requests resolved during the *parallel access* stage. As partitioned multicast sends the memory request to nine banks in parallel during its first

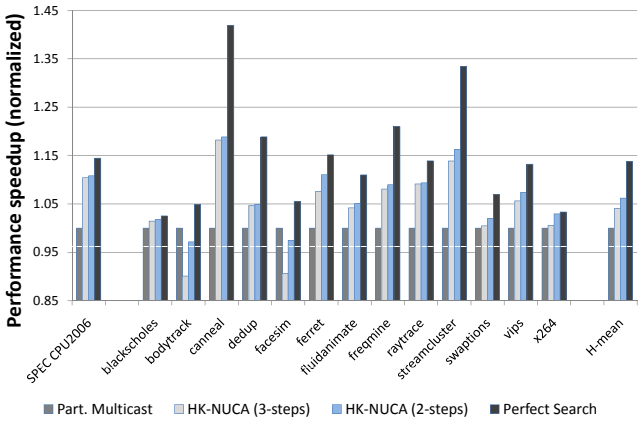


Figure 7. Performance results.

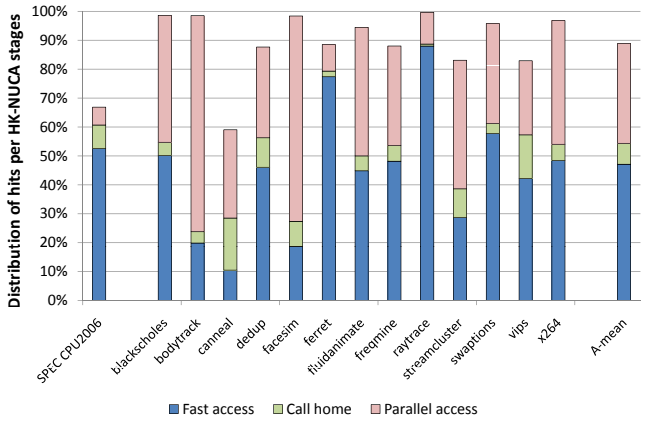


Figure 8. Distribution in HK-NUCA stages.

	SPEC CPU2006	blackscholes	bodytrack	canneal	dedup	facesim	ferret	fluidanimate	freqmine	raytrace	streamcluster	swaptions	vips	x264	A-Mean
0 messages	0.94	0.38	0.22	0.12	0.73	0.22	1.10	0.81	0.32	0.48	0.21	0.52	0.36	0.45	0.41
1 message	4.84	33.01	23.66	2.79	3.58	8.75	6.77	1.85	4.98	28.17	10.00	38.04	4.01	4.55	12.80
2 messages	9.56	39.61	33.86	8.45	11.09	18.14	14.31	3.76	15.16	19.99	18.96	36.90	12.82	13.28	19.68
3 messages	14.49	12.13	25.04	16.34	21.33	21.19	17.92	14.35	19.97	19.69	20.68	16.78	20.63	22.96	19.29
4 messages	18.27	14.60	12.10	21.87	25.93	21.80	17.63	24.19	22.13	15.94	12.05	5.90	21.93	31.54	18.83
5 messages	18.69	0.25	4.04	21.00	20.53	16.18	14.77	22.96	17.38	7.00	10.13	1.26	18.64	18.02	13.06
6 messages	15.24	0.03	0.86	15.51	10.93	9.30	10.93	18.82	11.78	3.44	9.52	0.29	12.64	6.26	8.38
7 messages	9.92	0.00	0.15	8.67	4.31	3.51	7.93	9.92	5.32	2.40	8.20	0.20	6.32	2.15	4.51
8 messages	5.13	0.00	0.05	3.67	1.23	0.83	4.98	2.43	2.03	1.97	5.59	0.09	2.11	0.62	1.97
9 messages	2.09	0.00	0.02	1.21	0.28	0.07	2.50	0.69	0.67	0.87	2.98	0.02	0.47	0.14	0.76
10 messages	0.66	0.00	0.00	0.30	0.06	0.01	0.88	0.18	0.18	0.04	1.22	0.00	0.07	0.02	0.24
11 messages	0.15	0.00	0.00	0.06	0.01	0.00	0.23	0.03	0.05	0.00	0.37	0.00	0.00	0.00	0.06
12 messages	0.02	0.00	0.00	0.01	0.00	0.00	0.04	0.00	0.01	0.00	0.08	0.00	0.00	0.00	0.01
13 messages	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00
14 messages	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Values in percentage (%)

Table III
NUMBER OF MESSAGES SENT DURING THE PARALLEL ACCESS STAGE OF HK-NUCA.

stage, it resolves memory requests faster than HK-NUCA when the requested data is in the central banks. HK-NUCA, however, would satisfy these request during the second or the third step of the algorithm, depending on the version of HK-NUCA assumed. Besides, the whole working set fits in the NUCA cache, so HK-NUCA can not benefit of the reduction on the miss resolution time. Fortunately, the trend on the applications for future processors follows the characteristics of the first scenario: applications with large working set or multiple applications running simulatenously.

The effectiveness of HK-NUCA relies on the accuracy of the HK-PTRs, in other words, the number of messages required to resolve a memory request during the *parallel access* stage. Table III shows that, on average, 85% of memory request resolved during the *parallel access* stage required up to 5 messages, which is far away from the most pessimistic situation: 14 messages. We also highlight that the accuracy of HK-NUCA remains constant with all simulated workloads, although their memory characteristics are very different.

B. Energy consumption analysis

HK-PTRs are the key structures that allow HK-NUCA for significantly reducing the number of messages required to resolve memory requests. However, HK-NUCA also introduces extra notification messages into the on-chip network to keep HK-PTRs updated. Figure 9 quantifies the on-chip network contention found. We find that HK-NUCA

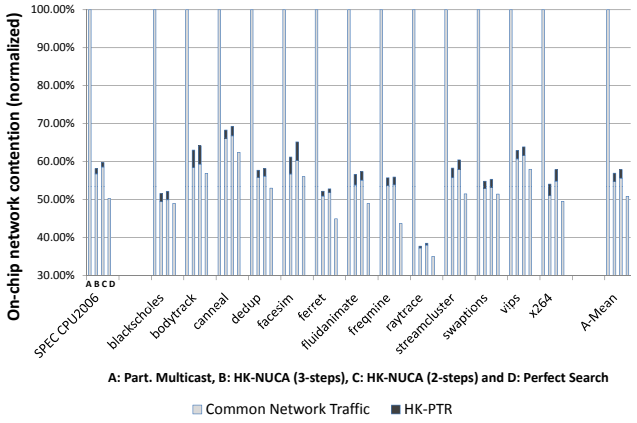


Figure 9. Quantification of the on-chip network traffic.

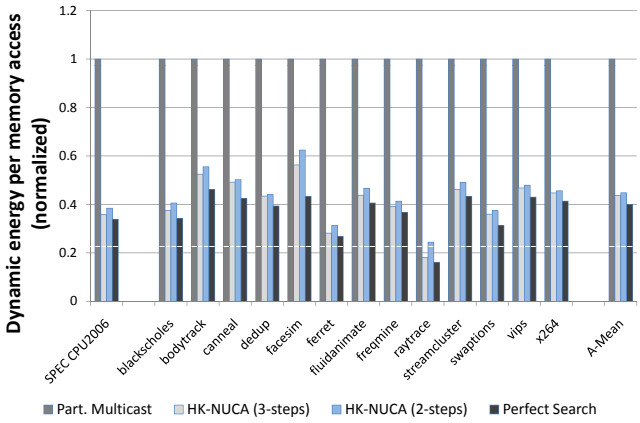


Figure 10. Dynamic energy per memory access.

	SPEC CPU2006	blackscholes	bodytrack	canneal	dedup	facesim	ferret	fluidanimate	freqmine	raytrace	streamcluster	swaptions	vips	x264	A-Mean
Part. multicast	11.24	9.82	9.49	11.66	9.99	9.87	9.95	10.33	10.06	9.11	11.04	9.53	10.66	10.12	10.03
HK-NUCA (3-steps)	3.67	2.61	3.78	5.53	3.91	4.53	2.49	4.13	3.86	1.51	4.57	2.63	4.09	3.52	3.82
HK-NUCA (2-steps)	4.09	2.95	3.85	5.57	4.03	4.65	3.05	4.51	4.09	2.33	4.71	2.86	4.19	3.77	4.06
Perfect Search	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table IV
AVERAGE NUMBER OF BANKS ACCESSED PER MEMORY REQUEST.

substantially reduces traffic on the on-chip network (on average, by 43%) compared to the partitioned multicast approach. Moreover, Figure 9 also shows that the messages introduced for updating HK-PTRs are less than 2.3% of the on-chip network traffic.

In order to resolve a memory request, and thanks to HK-PTRs that indicate which bank could potentially have the requested data, HK-NUCA does not have to access all the NUCA banks in which the requested data block might be mapped within the NUCA cache. Table IV shows the average number of banks accessed per memory request. We found that partitioned multicast, on average, accesses 10 banks per each memory request. Perfect search, which is the optimal search approach, requires to access one bank. HK-NUCA, however, requires, on average, 3.82 accesses per memory request assuming the 3-step version of the data search algorithm, and 4.06 accesses if the better-performing 2-steps version of HK-NUCA is assumed.

Figure 10 shows the dynamic energy consumed by the memory system compared to the partitioned multicast access scheme. HK-NUCA uses mechanisms to reduce the number of messages required to find a data block within the NUCA cache (fast access stage and HK-PTRs) which make it energy efficient. In particular, HK-NUCA reduces dynamic energy consumed per each memory access by more than 40%. On the other hand, because HK-NUCA hardware requirements (less than 0.4% hardware overhead, see Section IV-C), static energy is slightly increased, however, this is clearly compensated by the considerable reduction achieved in dynamic energy.

VII. Related Work

The increasing influence of wire delay in cache design means that access latencies to the last-level cache banks are no longer constant [1]. To address this problem, Kim et al. [2] introduced the Non-Uniform Cache Architecture (NUCA). A NUCA divide 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. Since then, there have been several studies using NUCA architectures in the literature that focuses on the policies that they manage: *placement* [21], [9], [7], [4], [8], *migration* [22], [23], *replacement* [24], [25], [26] and *access* [5], [27], [6].

The introduction of CMP architectures brought additional challenges to the NUCA architecture. Beckmann and Wood

[3] 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 perfect search mechanism that was difficult to implement. Kim et al. [2] were the first to point out the importance of the bank searching algorithm in a dynamic NUCA architecture. Although migration movements leverages D-NUCA to potentially outperform S-NUCA, the D-NUCA benefit is significantly diminished by the quality of the bank-searching algorithm within the cache. Huh et al. [4] also came to the same conclusion after analysing several alternatives to NUCA organisations. They concluded that, although D-NUCA performance potential dramatically outperforms that of other mechanisms, the benefits currently offered by D-NUCA organization do not justify the complexity of the design. Therefore, access policy becomes a key issue in NUCA designs.

Kim et al. [2] analysed two distinct searching policies: *incremental search* and *multicast search*. *Incremental search* searches for data in banks sequentially from the closest to the furthest bank, whereas *multicast search* accesses all NUCA banks in parallel. They also analysed two hybrid approaches that combined both incremental and multicast algorithms. One of them was *partitioned multicast* which is the two-phase multicast search that we used as baseline access scheme (see Section II). The other hybrid scheme that they proposed was *limited multicast* that searches data in parallel in a subset of banks from the bankset, and then sequentially in the rest of them. However, hybrid approaches that try to keep the best of both worlds are still far from ideal. One orthogonal way to improve any of the searching algorithms is to introduce tag information replication along the NUCA cache [9], [4], [23], [6]. Unfortunately, these approaches lead to increased access time, energy consumption and die area.

A novel bank access approach which predicts the bank in which data are most likely to be located has been proposed by Akioka et al. [5]. They divided the NUCA cache into *rings* to denote a set of banks that exhibit the same access latency and introduced a Last Access Based (LAB) scheme. This approach first accesses the ring that satisfied the previous request for the same data. They showed that, compared with parallel and serial access, LAB reduces energy consumption while maintaining similar performance. However, an accurate implementation is not addressed in terms of die area and access time. Ricci et al. [6] also dealed with prediction by identifying cache bank candidates with high accuracy through Bloom filters. The underlying idea is to keep approximate information about the contents of banks. Preliminary results show that compared to a simple D-NUCA approach the number of messages inside the cache network decreases while the hit ratio accuracy is maintained. Unfortunately, the work focuses on demonstrating the potential and an implementation of the approach is not presented, nor is an analysis of performance or energy consumption.

Recent works have proposed NUCA designs based on S-NUCA, but redefining placement decisions to be able to locate data in multiple banks within the NUCA cache [7], [28]. These designs intend to have the best of the two worlds, locality from a D-NUCA design and the simplicity found in S-NUCA. However, they rely on complex OS-assisted data placement schemes.

Finally, Muralimanohar et al. [27] focused on the network on-chip design to alleviate the interconnect delay bottleneck. They proposed a NUCA approach that uses two different physical wires to build NUCA architectures (one of these wires provided lower latency and the other provided wider bandwidth). Then, they proposed two different bank searching algorithms (early and aggressive lookup) that benefits from this novel interconnection design.

VIII. Conclusions

Although exploiting flexible mapping D-NUCA promises significant benefits, those benefits are restricted by the access algorithm used. Previous works agree that the simplest design is probably the best: S-NUCA [3], [4], [2]. However, D-NUCA still holds promise and it is worth waiting for an efficient access scheme that allows it to make the promised benefits a reality. This is HK-NUCA. It does not only improve performance, but also energy consumption. On one hand, by taking advantage of migration movements, the *fast access* stage of HK-NUCA provides high-performance at a minimum cost. *Home knowledge*, on the other hand, allows memory requests to be satisfied by only accessing the NUCA banks that could potentially have the requested data. This means that HK-NUCA significantly reduces on-chip network traffic and diminishes router delays due to congestion. On average, HK-NUCA outperforms previous access schemes for D-NUCA caches by 6%, and reduces the dynamic energy consumed by the memory system by 40%.

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.

References

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

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

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

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

[5] 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 *Procs. of the International Conference on Computer Design*, 2008.

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

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

[8] J. Merino, V. Puente, and J. A. Gregorio, "Sp-nuca: A cost effective dynamic non-uniform cache architecture," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 2, pp. 64–71, May 2008.

[9] M. Hammoud, S. Cho, and R. Melhem, "Acm: An efficient approach for managing shared caches in chip multiprocessors," in *Procs. of the 4th Intl. Conference on High Performance and Embedded Architectures*, 2009.

[10] 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.

[11] 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*, 2005.

[12] D. Molka, D. Hackenberg, R. Schone, and M. S. Muller, "Memory performance and cache coherency effects on an intel nehalem multiprocessor system," in *Procs. of the International Conference on Parallel Architectures and Compilation Techniques*.

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

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

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

[16] 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.

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

[18] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks," in *Procs. of the 35th International Symposium on Microarchitecture*, 2002.

[19] Micron, "System power calculator," in *http://www.micron.com/*, 2009.

[20] E. Grochowski, R. Ronen, J. Shen, and H. Wang, "Best of both latency and throughput," in *Procs. of the 22nd Intl. Conference on Computer Design*, 2004.

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

[22] M. Hammoud, S. Cho, and R. Melhem, "Dynamic cache clustering for chip multiprocessors," in *Procs. of the Intl. Conference on Supercomputing*, 2009.

[23] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son, "A novel migration-based nuca design for chip multiprocessors," in *Procs. of the International Conference on Supercomputing*, 2008.

[24] J. Lira, C. Molina, and A. González, "Last bank: dealing with address reuse in non-uniform cache architecture for cmps," in *Procs. of the International Conference on Parallel and Distributed Computing*, 2009.

[25] ——, "Lru-pea: A smart replacement policy for non-uniform cache architectures on chip multiprocessors," in *Procs. of the International Conference on Computer Design*, 2009.

[26] ——, "The auction: Optimizing banks usage in non-uniform cache architectures," in *Procs. of the International Conference on Supercomputing*, 2010.

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

[28] M. Chaudhuri, "Pagenuca: Selected policies for page-grain locality management in large shared chip-multiprocessor caches," in *Procs. of the 15th IEEE Symposium on High-Performance Computer Architecture*, 2009.