Performance Analysis of Non-Uniform Cache Architecture Policies for Chip-Multiprocessors Using the Parsec v2.0 Benchmark Suite

Javier Lira [1] Carlos Molina [2] and Antonio González [3] [4]

Abstract— *Non-Uniform Cache Architectures (NUCA) have been proposed as a solution to overcome wire delays that will dominate on-chip latencies in Chip Multiprocessor designs in the near future. This novel means of organization divides the total memory area into a set of banks that provides non-uniform access latencies and thus faster access to those banks that are close to the processor. A NUCA model can be characterized according to the four policies that determine its behavior: bank placement, bank access, bank migration and bank replacement. Placement determines the first location of data, access defines the searching algorithm across the banks, migration decides data movements inside the memory and replacement deals with the evicted data.*

This paper analyzes the performance of several alternatives that can be considered for each of these four policies. Moreover, the Parsec v2.0 benchmark suite has been used to handle this evaluation because it is a representative group of upcoming shared-memory programs for Chip Multiprocessors. The results may help researchers to identify key features of NUCA organizations and to open up new areas of investigation.

I. Introduction

The continuing technological advances in the scale of integration have ensured that the number of transistors that can be integrated into a single chip will double every two years. This prediction, known as Moore's Law [1], has been in place 40 years and it is widely accepted that this trend will continue over the next 10-15 years. Therefore, future processors will have billions of tiny transistors. Against this background, an important question that arises is how current processors can efficiently use this technology.

Chip Multiprocessors (CMPs) have emerged as a dominant paradigm in system design [2], [3]. Several commercial microprocessors are beginning to include multiple cores (2 to 8, depending on the model) with a shared cache. Moreover, as we increase the scale of integration, the chips include more and more cores, which could lead to 64 processor cores being placed on a chip by the middle of the next decade.These multicore systems incorporate larger and shared second-level caches with a homogeneous access time. However, traditional cache architectures assume that each level in the cache hierarchy has a single and uniform cache access time, but the increasing communication delay causes the hit time of large on-chip caches to be a function of a line's physical location within the cache. Consequently, cache access time becomes a continuum of latencies rather than a single discrete latency [4], [5]. Non-Uniform Cache Architecture (NUCA), that was first proposed by Kim et al [6], exploits this non-uniformity to provide master access to cache lines in those portions of the cache that are closer to the processor.

The underlying concept behind a NUCA system involves dividing the whole cache into smaller banks. Each of these banks traditionally has a single discrete latency, although this is much smaller than it would be if the whole cache was a uniform cache. Data are distributed among all the banks, so the total latency for getting a single piece of data from a processor includes the requesting and the responding routing time from the processor to the bank containing the requested data plus the latency of the bank. A NUCA model can be characterized by the following four policies that are involved in its behavior: *Bank Placement Policy, Bank Access Policy, Bank Migration Policy*, and *Bank Replacement Policy.*

This paper aims to analyze how NUCA performs on a Chip Multiprocessor using the Parsec v2.0 benchmark suite. Starting from a base configuration, it attempts to show the potential of each of the four policies that characterizes the behavior in a NUCA system. In this way, several alternatives for each policy will be described, evaluated and discussed.

The remainder of this paper is structured as follows. Section II presents the baseline model that has been assumed, the simulation tools and a brief description of the benchmarks used. Section III describes the alternatives of each bank policy considered in this study. Section IV presents the results obtained during the simulations. Related work is summarized in Section V. Finally, Section VI outlines the main conclusions of this work.

II. Experimental Framework

A. Baseline Model

This paper deals with a L2 NUCA organization based on that proposed by Beckmann and Wood [7]. Figure 1 illustrates this baseline model. A die with 8 cores on the edges and a shared L2 NUCA cache in the center has been assumed. Each core maintains its own private first level cache that is divided for data and instructions. First level caches are 2-way set associative while L2 NUCA cache is 8 way set associative. The MOESI coherence protocol maintains correctness and robustness in the memory system. Moreover, the length of the wire that connects the NUCA with the third level of the

[1]Universitat Politècnica de Catalunya, `jlira@ac.upc.edu`
[2]Universitat Rovira i Virgili, `carlos.molina@urv.net`
[3]Intel Barcelona Research Center
[4]Intel Labs - UPC, `antonio.gonzalez@intel.com`

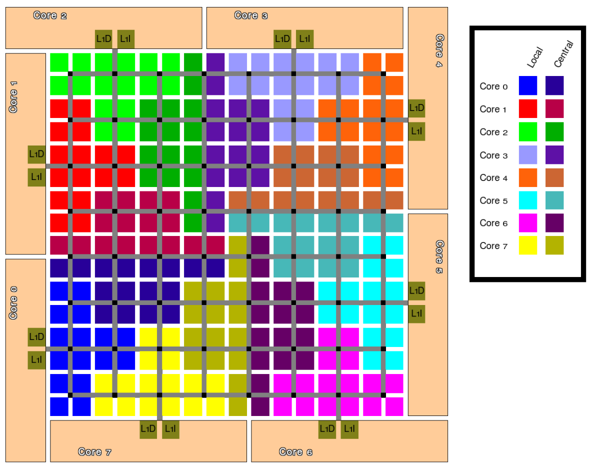

Fig. 1. Organization of NUCA architecture.

memory hierarchy is the same for all cores and this wire comes from the center of the NUCA structure.

The NUCA cache is divided into 256 smaller banks structured in a 16x16 mesh connected via a 2D interconnection network. In Figure 1, the NUCA banks are represented by colored squares and the interconnection network is represented by grey lines (wires) and black points (network switches). The NUCA cache is shared among all cores. There are 8 cores and each core owns 32 banks. These banks are classified into two groups, local and central, depending on their physical distance from their owner core. This means that each core has 16 local banks, 16 central banks and 224 distant banks.

Table I summarizes the CMP parameters assumed for the baseline model.

Processors	8, 4-way SMT
Branch Predictor	YAGS
Instr. Window / ROB	64 / 128 entries
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	8 KBytes
NUCA Bank Latency	4 cycles
Router Latency	1 cycle
Memory Latency	350 cycles (from core)

TABLE I
CMP PARAMETRIZATION.

B. Simulation Tools

We used Simics [8], a full system execution-driven simulator extended with the GEMS (General Execution-driven Multiprocessor Simulator) toolset [9], to characterize and simulate NUCA on a Chip Multiprocessor. We also used a couple of GEMS modules, Ruby and Opal, which improve CMP functionalities. Ruby is a highly accurate timing model, whereas Opal is an extension for supporting out of order execution. GEMS provides detailed simulation of multiprocessor systems and it makes processors deal with SPARCv9 ISA. Finally, a Solaris v10 operating system has been installed on the emulated machine.

A CACTI 6.0 tool [10] greatly assists circuit simulation. It estimates area, access time and power dissipation of on-chip cache organizations. Assuming a 45nm technology, we modeled the baseline NUCA memory to determine the parameters of the cache access time.

C. PARSEC v2.0 Benchmark Suite

The Princeton Application Repository for Shared-Memory Computers (PARSEC v2.0) has been recently released [11]. This benchmark suite contains emerging applications and commercial solutions that cover a wide area of working sets and allow current Chip Multiprocessor technologies to be studied more effectively [12]. In this paper we evaluate the whole set of the PARSEC v2.0 benchmark suite with the *simlarge* input data sets. Moreover, we forward a significant number of instructions for preventing initialization behavior and thread creation. Then, we fast-forward while warming all caches for 500 million cycles, and finally, we collect the statistics for the following 200 million cycles.

III. Bank Policy Approaches

This section describes the alternatives of each NUCA policy that will be further simulated and analyzed.

A. Bank Placement Policies

This policy determines where a data element should be placed in the NUCA cache memory when it comes from the off-chip memory or from other caches. It also determines in which set of banks this data can be located during its life. A typical configuration will allow placing data in some banks of the NUCA caches. Notice that placing a data block in any of the banks could require a non-affordable search algorithm, whereas restricting data to just one bank will minimize the benefit of achieving the lowest access time for those data blocks that are being accessed more frequently. Therefore, three alternatives are considered:

- **1B + Static:** The data is always placed in the same bank and this depends on the address bits of the data block.
- **16B + Static:** A data block can be located in 16 banks during its life in the NUCA cache, that is, in one of the local or central banks of each of the eight cores in our CMP architecture. The initial location of incoming data blocks from the off-chip memory is determined statically among the 16 banks and based on the address bits of the data block. On the other hand, an incoming data block from an L1 cache replacement is always placed in the corresponding local bank of the core that produced that eviction.

- **16B + Local:** As in the previous approach, a data block will reside in just 16 banks corresponding to one local and one central bank of each of the 8 cores. The difference is that the initial location will always be the local bank of the core that produces the eviction, regardless of whether the data block comes from the L1 cache or the off-chip memory.

B. Bank Access Policy

This policy determines how to search for a data block among the banks in which it is located. This may involve a serial search, a parallel search or a combination of both. Notice that serial search will reduce the number of bank lookups but will increase the miss resolution time. On the other hand, parallel search reduces miss resolution time but increases the number of bank lookups and thus, network traffic and energy consumed. We evaluate the following approaches:

- **Serial:** This is the most simple access approach. It consists on accesing one by one all the suitable banks where data can be located, until the requested data is found or a miss is identified. It saves network traffic and energy but miss resolution time is far from optimal.
- **Parallel:** This algorithm searches in parallel in all the banks where data can be located. This approach is not affordable in terms of energy and collapse of the network, but it does obtain the lowest miss resolution time possible.
- **Mixed:** This approach combines the serial and parallel searches but giving more weight to parallel access. Assuming for instance that data can be located in 16 banks (one local and one central of each core), the searching algorithm is divided in two steps. The first step searches in parallel the local bank of the requesting core and all the central ones (9 banks in total). If the data block is not found, the second step is triggered to simultaneously search the 7 remaining banks (local to other cores).

In addition, a *Perfect* searching scheme is also considered to avoid the effect of *Bank Access* policy when analyzing *Bank Placement, Bank Migration* and *Bank Replacement* policies. This non-affordable approach knows exactly where data is located (off-chip or in a specific bank) and, therefore, it returns minimum access latency.

C. Bank Migration Policy

This policy determines if a data element is allowed to change its placement from one bank to another bank of the NUCA cache memory. It also defines which data should be migrated, when this data should be migrated and to which bank it should be moved.This will place the most frequently accessed data as close as possible to each core. The following alternatives are considered:

- **Static:** There is no migration assumed in this static approach. Therefore, when data is first located in its corresponding bank, it will stay there forever until it is replaced.
- **Dynamic + Swapping:** This dynamic approach assumes a gradual migration mechanism that moves data to the local bank of the requesting core. This migration policy is applied just after the core has accessed a data block inside the NUCA cache. Assuming for instance that data can be located in 16 banks (one local and one central of each core), the migration algorithm promotes the data to a bank that is one-step closer to the processor that has just accessed it. This movement, however, may cause swapping between two banks in the NUCA cache. Therefore, data from the destination bank is moved one-step further from the processor. The data accessed is *gradually* promoted as follows: *local bank* (remote) *central bank* (remote) *central bank* (requestor) *local bank* (requestor). Thus, data located in the bank farthest from the processor that has just accessed it, requires three accesses in order to be placed in the closest bank.
- **Dynamic + Replication:** This approach behaves like the previous one with the difference that data replication is allowed. Thus, data is replicated instead of swapped in case the line permissions state is read-only. Notice that this behavior tries to prevent two cores from constantly competing for the same data.

D. Bank Replacement Policy

This policy determines how NUCA cache behaves when there is a data eviction from one of the banks. In this case, the Least Recently Used (LRU) data block within the same bank and cache-way, where the incoming data would enter, is evicted from the NUCA cache bank. Bank replacement policy determines what to do with the evicted data, and the following approaches are considered for this policy:

- **Zero Copy:** This approach assumes that an evicted data element is sent back to the off-chip memory.
- **One Copy:** Instead of sending the evicted data to the off-chip memory, this approach reallocates data in another lower-priority bank further from the processor. Thus, the evicted data that comes from a local bank is reallocated to a central bank in the same core that produced the eviction. If the evicted data comes from a central bank, it is sent back to the off-chip memory.
- **Last Bank:** An additional bank [13] can be added to store all the evicted data from the NUCA cache. In this case, this bank behaves as a victim cache [14] instead of sending data to the off-chip memory when replacement. Furthermore, when a hit occurs on the *Last Bank*, the data is moved to the regular NUCA cache.

IV. Analysis of Results

In this section we evaluate the alternatives of one NUCA bank policy assuming a fixed configuration for the remaining polices. Therefore, the following approaches are assumed as a baseline configuration for each of the 4 policies that determine the behavior of a NUCA cache.

- Bank Placement Policy: **16B + Static**
- Bank Access Policy: **Perfect**
- Bank Migration Policy: **Dyn + Swap**
- Bank Replacement Policy: **Zero Copy**

A. Bank Placement Policy

We analyze placement restricted to a single bank and placement allowed in a set of banks (*16B Static* and *16B Local*). The former does not apply all baseline configuration approaches because access is restricted to one bank and migration is not possible. The latter approach assumes that a data block can be located in 16 banks during its life in the NUCA. *16B Static* places data according to the address bits, whereas *16B Local* always places data on the local bank of the core that produces the eviction.

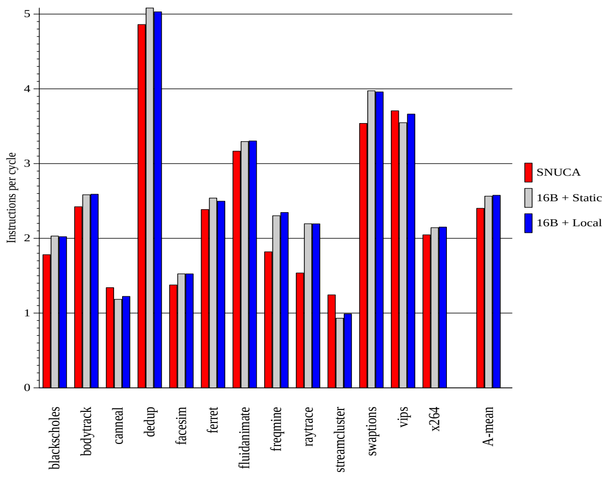

Fig. 2. IPC of bank placement policy alternatives.

Figure 2 shows the performance potential of the presented approaches. We observe that there are no significant differences between *16B Static* and *16B Local* approaches. We also observe that *16B* approaches outperform *1B Static*, but this improvement is not as high as expected. Note that *1B Static* does not allow data to be placed close to requesting cores, meanwhile *16B* approaches provide data migration support. On the other side, *1B Static* spread data over the whole NUCA cache fairer than *16B* approaches in which migrations concentrate data in a few banks. We believe that a correlated exploration of bank placement and bank migration policies will provide better performance results when a set of banks is considered to locate a data.

B. Bank Access Policy

In this policy we analyze two opposite approaches (serial and parallel access). There is a trade-off between both opposite alternatives when considering performance in front of network traffic and energy consumption. *Serial* access saves traffic and energy but provides higher access time. A *Parallel* search in bank access policy provides the lowest access time but seriously increases the traffic contention and energy used since more banks must be accessed.

Figure 3 shows the performance potential of parallel weighted approaches. They heavily outperform the serial approach. In particular, *Parallel* and *Mixed* accesses achieve around an average IPC of 2.55, whereas *Serial* access achieves an average IPC close to 2.15. These results suggest the suitability of a parallel access approach but, further energy consumption analysis must be done to find a good combination of performance and energy.

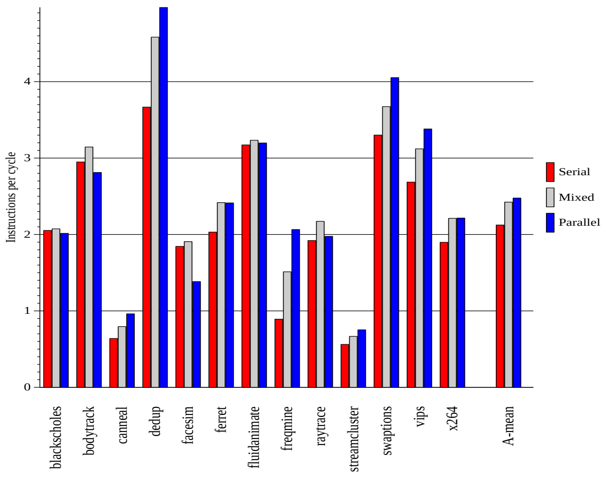

Fig. 3. IPC of bank access policy alternatives.

C. Bank Migration Policy

This policy determines how movements among banks can affect the performance of a NUCA cache. We compare no migration with a dynamic migration mechanism that moves data to the local bank of the requesting core. Furthermore, dynamic migration may or may not replicate data.

Figure 4 shows the performance potential of a migration policy. We observe that migration performs well in all cases. Therefore, a non-migration approach does not seem to be a good choice when only performance is considered. As outlined in Section IV-A, we believe that there is a high correlation between the bank placement and bank migration policies. Therefore, further analysis will be needed to combine both policies.

D. Bank Replacement Policy

We now evaluate several bank replacement policy approaches. As explained before, *Zero Copy* sends the evicted data back to the off-chip memory, *One Copy* places data in another bank and *Last Bank* assumes there is an additional bank which has the same size as the rest. This bank has been located on the centre of chip at the same distance from all cores acting as the last level cache between the NUCA cache and the off-chip memory. We also assume

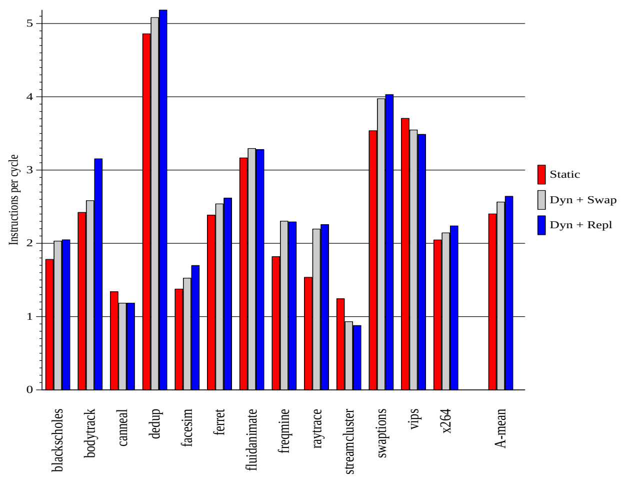

Fig. 4. IPC of bank migration policy alternatives.

a huge non-affordable *Last Bank* of 16 MBytes to estimate the potential benefit of this mechanism. Figure 5 shows the IPC results for each alternative.

Generally, *One Copy* outperforms the *Zero Copy* and *Last Bank* configurations. On average, *One Copy* achieves 2.55 of IPC whereas *Zero Copy* and *Last Bank* both achieve around 2.45 of IPC. The figure also shows promising performance potential when an unbounded *Last Bank* configuration is assumed. Thus, a better exploration of an affordable last bank approach is needed.

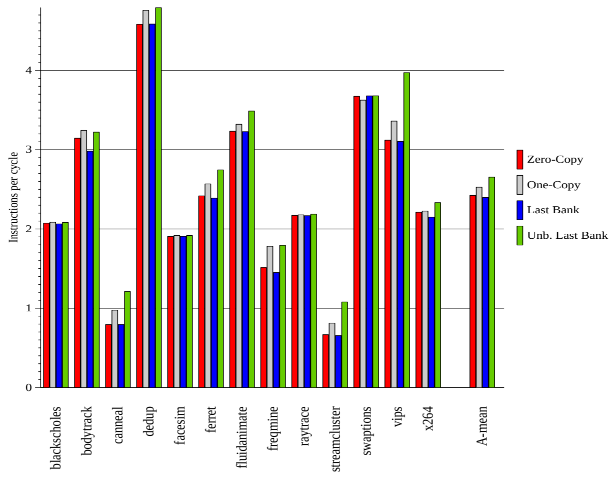

Fig. 5. IPC of bank replacement policy alternatives.

V. Related Work

Kim et al. [6] introduced the concept of Non-Uniform Cache Architecture (NUCA) and designed several NUCA caches by partitioning the cache into multiple banks and using a switched network to connect these banks. Two main placement alternatives have been proposed: Static NUCA (S-NUCA) and Dynamic NUCA (D-NUCA). 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 master banks. Furthermore, two alternative bank replacement policy are also proposed: *zero-copy* and *one-copy*.

Huh et al. [15] analyzed placement by introducing the concept of the sharing degree in a NUCA bank. The sharing degree is the number of cores that share a specific bank, so a sharing degree of one signifies a private cache. Larger sharing degrees reduce the number of misses, thus optimizing the cache capacity usage. Unfortunately, smaller sharing degrees reduce hit latencies. Hardavellas et al [16] observed that cache accesses for instructions, shared data and private data exhibit different characteristics. Therefore, they proposed a mechanism (Reactive-NUCA) that applies different migration and placement policies depending on type of data. Finally, Hammoud et al [17] introduced a distributed cache management scheme that monitors the behaviour of the program and makes related placement decisions.

Migration was considered by Beckmann and Wood [7]. They gathered with current proposals for managing wire delays and combined them with CMPs. They also demonstrated that block migration is less effective for CMP because 40-60% of hits in commercial workloads were satisfied in the central banks. However, to improve CMP performance, the capability of block migration relied on a smart search mechanism that was difficult to implement. Kandemir et al [18],[19] proposed a migration mechanism that tries to select the most appropriate locations to place data shared by multiple cores. Their proposal was based on modelling the two-dimensional post office placement problem. Another migration scheme was proposed by Hammoud et al [20] to move data to banks that best minimize the average NUCA access latency. They predicted the optimal location of data by monitoring the behaviour of programs.

Muralimanohar et al [21] proposed a different approach in NUCA architectures. These authors proposed the use of two different physical wires to build NUCA architectures. One of these wires provided lower latency and the other higher provided bandwidth. They then proposed two different bank searching algorithms [21]. A novel bank access approach has been proposed by Akioka et al [22]. They proposed a mechanism that predicts the bank in which data is most likely to be located achieving an energy compared to parallel access with a performance similar to serial access. Ricci et al [23] proposed a smart lookup mechanism for NUCA that deals with Bloom filters.

Lira et al [13],[24] analyzed several replacement alternatives and studied the characterization of those data that are evicted and then reinserted in the future. Furthermore, they also proposed adding an extra bank (called *Last Bank*), located in the middle of the NUCA cache at the same distance to all cores, which collects all the evicted data. Finally, they proposed filtering evicted data so that the *Last Bank* only contains those data blocks, which are most likely to be accessed in the near future.

VI. Conclusions and future work

We analyse how NUCA organizations perform according to different approaches, each using the four policies that characterize their behavior (placement, access, migration and replacement). The Parsec v2.0 benchmark suite has been assumed in all the simulations. Results show us that there is still room for improvement in all policies.

Bank placement policy analysis shows that assuming a subset of banks for the whole cache is more desirable, as data needs to be promoted to closer and faster banks. However, restricting placement to a single bank provides significant performance results. On the other hand, believing that data can be placed in the whole set of banks may lead to non-affordable access and migration policies.

Parallel searches in bank access policy provide the lowest access time but also significantly increase energy consumption and can collapse the interconnection network, since all banks have to be accessed. Therefore, a more affordable mechanism that combines parallel and serial searches is required.

NUCA organizations benefit by placing the most frequently accessed data close to the requestor processor. Thus, migration is a key bank policy in terms of performance improvement as results show significant slowdown when no migration is assumed. On the other hand, although no important differences have been observed between the gradual approaches, we believe that further exploration of this policy will introduce significant improvements.

Finally, bank replacement policy does not seem to cause huge differences among the alternatives analyzed except when the unbounded last bank is considered. This result suggests that new approaches need to be explored for replacement policy.

VII. Acknowledgements

This work is supported by the Spanish Ministry of Science and Innovation (MCI) and FEDER funds of the EU under contracts TIN 2007-61763 and TIN2007-68050-C03-03, the Generalitat de Catalunya under grant 2005SGR00950, and Intel Corporation. Javier Lira is supported by the MCI under FPI grant BES-2008-003177.

References

[1] G. E. Moore, *The future of integrated electronics*, In Fairchild Semiconductor internal publication, 1964.

[2] P. Frost Gorder, "Multicore processors for science and engineering," *Computing in Science & Engineering*, March-April 2007.

[3] R. Low, "Microprocessor trends: multicore, memory, and power developments," *Embedded Computing Design*, September 2005.

[4] 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.

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

[6] 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.

[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] 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*, vol. 35-2, pp. 50–58, Computer, 2002.

[9] 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*, Sept. 2005.

[10] 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.

[11] 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.

[12] C. Bienia, S. Kumar, and K. Li, "Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Proceedings of the IEEE International Symposium on Workload Characterization*, 2008.

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

[14] 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.

[15] 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.

[16] 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*, 2009.

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

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

[19] M. Kandemir, F. Li, M. J. Irwin, and S. W. Son, "A novel migration-based nuca design for chip multiprocessors," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2008.

[20] 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*, 2009.

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

[22] 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.

[23] 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*, 2006.

[24] J. Lira, "Data replacement policy on non-uniform cache architectures for chip multi-processors," M.S. thesis, Universitat Politécnica de Catalunya, 2008.