Last Bank: Dealing with Address Reuse in Non-Uniform Cache Architecture for CMPs

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

[1] Universitat Politécnica de Catalunya
[2] Universitat Rovira i Virgili
[3] Intel Barcelona Research Center, Intel Labs - UPC

Abstract. *In response to the constant increase in wire delays, Non-Uniform Cache Architecture (NUCA) has been introduced as an effective memory model for dealing with growing memory latencies. This architecture divides a large memory cache into smaller banks that can be accessed independently. Banks close to the cache controller therefore have a faster response time than banks located farther away from it. In this paper, we propose and analyse the insertion of an additional bank into the NUCA cache. This is called* Last Bank. *This extra bank deals with data blocks that have been evicted from the other banks in the NUCA cache. Furthermore, we analyse the behaviour of the cache line replacements done in the NUCA cache and propose two optimisations of Last Bank that provide significant performance benefits without incurring unaffordable implementation costs.*

1 Introduction

Advances in technology have made it possible for more and more transistors to be placed on a single chip. This enables computer architects to deal with a huge number of transistors and build smarter processors. In the 1990s, the main trend in constructing computers to satisfy the high performance requirements of different applications was to increase clock speed and introduce even greater complexity in order to exploit Instruction-Level Parallelism (ILP) in applications. While increasing clock speed improves the performance of the processors, it also increases power consumption. At the end of the 1990s, power efficiency became a critical issue because the power consumption per unit of area was growing dramatically and almost reaching the limits of affordability. Another disadvantage of increasing clock speed is that, as memory frequency does not grow as fast as processor frequency, the gap between processor speed and memory speed has widened dramatically.

Today it is feasible to integrate more than one processor on a single chip. This has opened up a new era in computer architecture (called the *Multicore era*) in which the smarter integration of more than one core on a single chip becomes the new challenge [14, 9]. Chip Multiprocessors (CMPs) can work at much lower clock frequencies than single core processors, thus solving the power consumption problem. CMPs also moderate the effects of the broad gap between processor and memory speed by exploiting Thread-Level Parallelism (TLP). However, new issues arise from this novel architecture. Chip complexity has increased with the introduction of multiple processors. A high performance on-chip interconnection network is required to connect all the components on the die. One of the main advantages of CMPs over other multiprocessors is the on-chip network, which enables data to be shared at very low latency.

The memory subsystem is also more complex in CMPs [20]. Access to the main memory is very expensive due to main memory response time and network delay. Having a shared last-level cache on-chip is therefore a key issue in this kind of architecture. Today's CMP architectures incorporate larger and more complex cache hierarchies. Recent studies have proposed mechanisms for dealing with new challenges to the memory system posed by CMP architectures, some of the most notable of these being cooperative caching [6, 5], victim replication [21], adaptive selective replication [2] and other works that exploit the private/shared cache partitioning scheme [8, 10].

The increasing influence of wire delay in cache design means that access latencies to the last-level cache banks are no longer constant [1, 17]. Non-Uniform Cache Architectures (NUCAs) have been proposed [13] as a way of addressing 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 internal wires of the cache.

In this paper we propose and analyse the insertion of an additional bank into the NUCA cache. This bank, called *Last Bank*, acts as a last-level cache on the chip by dealing with data blocks evicted from the NUCA cache. Moreover, we propose two optimisations for the Last Bank mechanism: *Selective Last Bank* and *LRU prioritising Last Bank*.

The remainder of this paper is structured as follows. Section 2 describes the baseline architecture assumed in our studies. Experimental methodology is presented in Section 3. Section 4 introduces the Last Bank mechanism. Section 5 analyses the NUCA cache line replacements and motivates the Last Bank optimisations proposed in Section 6. Our results are analysed in Section 7. Related work is discussed in Section 8, and concluding remarks are given in Section 9.

2 Baseline Model

As illustrated in Figure 1, the baseline architecture consists of an eight-processor CMP based on that of Beckmann and Wood [3]. 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 separated caches: one for instructions and one for data. The NUCA cache is then the second-level cache memory and it 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. The banks in the NUCA cache are also *logically* separated into 16 banksets that are either *local banks* (lightly shaded in

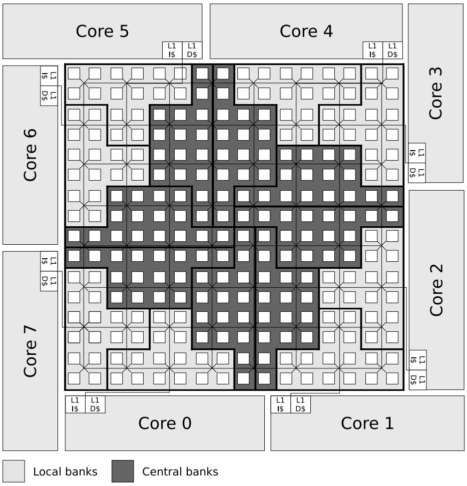


Fig. 1. Baseline architecture layout.

Figure 1) or *central banks* (darkly shaded in Figure 1) in accordance with their physical distance to the processors.

3 Experimental Framework

We used the full-system execution-driven simulator, Simics [15], extended with the GEMS toolset [16]. GEMS provides a detailed memory-system timing model that enabled us to model the NUCA cache architecture. The simulated architecture is structured as a single CMP made up of eight homogeneous cores. Each core is a superscalar out-of-order SPARCv9 processor modelled by the Opal simulator, which is one of the extensions that GEMS provides for Simics. With regard to memory hierarchy, each core provides a first-level cache, which is in fact two caches: one for instructions and one for data. The second level of the memory

Processors	8, 4-way SMT
Branch Predictor	YAGS
Instr. Window / ROB	64 / 128 entries
Block size	64 bytes
L1 Cache (Instr./Data)	8 KBytes, 4-way / 8 KBytes, 4-way
L2 Cache (NUCA)	1 MByte, 4-way, 256 Banks
NUCA Bank	4 KBytes, 4-way
L1 Latency	3 cycles
NUCA Bank Latency	2 cycles
Router Latency	1 cycle
Memory Latency	350 cycles (from core)

Table 1. Configuration parameters.

hierarchy is the NUCA cache. In order to maintain correctness and robustness in the memory system we used the MOESI token-based coherence protocol. Table 1 summarises the configuration parameters assumed in our studies. The access latencies of the memory components are based on models done with the CACTI 6.0 [19] modelling tool, which is the first version of CACTI that provides support for modelling NUCA caches.

We simulated a set of workloads from the PARSEC [4] benchmarks with *simlarge* input data sets. The method we used for the simulations consisted of, first skipping both the initialisation and thread creation phases, then fast-forwarding while warming all caches for 100 million instruction intervals, and finally performing a detailed simulation for 200 million instruction intervals.

4 Last Bank

Cache memories take advantage of the temporal and spatial data locality that applications usually exhibit. However, the whole working set does not usually fit into the cache memory, causing capacity and conflict misses. These misses mean that a line that may be accessed later has to leave the cache prematurely. As a result, evicted lines that are later reused return to the cache memory in a short period of time. This is more pronounced in a NUCA cache memory because data movements within the cache are allowed, so the most recently accessed data is concentrated in a few banks rather than spread over the entire cache memory. Therefore, we propose adding an extra bank to deal with data blocks that have been evicted from the NUCA cache. This extra bank, called *Last Bank*, provides evicted data blocks a second chance to come back to the NUCA cache without leaving the chip.

Last Bank, which is as large as a single bank in the NUCA cache, acts as the last-level cache between the NUCA cache and the off-chip memory. It is physically located in the centre of the chip at about the same distance to all cores. When there is a hit on Last Bank, the data block that is being accessed leaves Last Bank and goes back to the corresponding bank in the NUCA cache.

Figure 2 shows the performance gain achieved by adding Last Bank to the baseline configuration. On average, the Last Bank configuration outperforms the baseline with an IPC improvement of 4%. On the other hand, we also evaluated an unaffordable setup in which the evicted data blocks nearly always fit in Last Bank. The 64-MByte-sized Last Bank configuration shows the significant performance potential of this proposal, that is, an average IPC improvement of 22%. Thus, we conclude that although adding Last Bank to the baseline configuration results in performance benefits, these benefits are strictly limited by the size of Last Bank. The following sections analyse the behaviour of the evicted lines and propose two optimisations that provide extra IPC improvement to the Last Bank proposal without incurring unaffordable implementation costs.

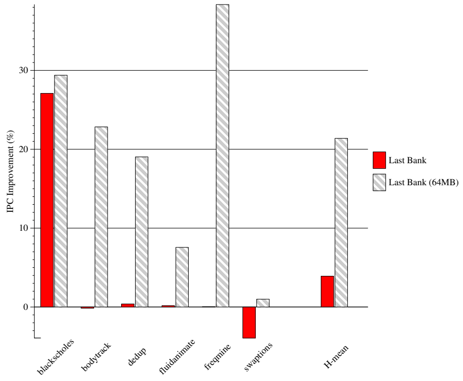


Fig. 2. Speed-up achieved with Last Bank compared to baseline configuration.

5 Characterization of NUCA cache line replacements

This section analyses the behaviour of the data blocks that leave the NUCA cache, focusing on those that later return. The aim of this study is to find hints to help us guess whether a data block that is being evicted from the NUCA cache will be accessed later.

5.1 Frequency of insertions into NUCA cache of reused addresses

We first analysed the frequency of insertions of reused addresses to find out whether most reinsertions are concentrated on a few addresses or whether the number of reinsertions are fairly spread out over all the reused addresses.

Figure 3(a) shows the insertion frequency of addresses that have been previously evicted for each of the workloads simulated (note that the X-axis deals with an exponential scale). The figure shows that 16% of reused addresses represent almost 50% of total reinsertions. This indicates that the assumed workloads generally concentrate a huge number of reinsertions on a few addresses. In this case, if we prevent this 16% of addresses from being evicted with a selective mechanism, off-chip requests would reduce dramatically as almost 50% of total reinsertions would become on-chip requests. Performance would therefore improve accordingly.

5.2 Time between eviction and insertion of a reused address

The second part of the analysis focused on the cycles passed between the eviction of a reused address and the reinsertion of the same address.

Figure 3(b) presents, for each workload, the percentage of addresses that return to the NUCA cache in a certain period of cycles. On average, nearly 30% of evicted addresses that are later inserted into the NUCA cache return in less than 100,000 cycles. In fact, with the *blackscholes* workload over 50% of evicted addresses return in less than 1,000 cycles. This clearly explains the benefit obtained by the Last Bank configuration.

5.3 Last location of evicted data blocks

This section analyses the location of evicted data blocks before replacement in order to evaluate whether the probability of an evicted data block being reinserted into the NUCA cache is related to the last NUCA bank in which it was stored. For the sake of simplicity, we have classified the NUCA banks according to the two bank categories introduced in Section 2 (*local* and *central*).

The results shown in Figure 3(c) are classified into the following three categories: 1) evicted data blocks that are not later accessed (0 R), 2) evicted data blocks that are later accessed only once more (1 R), and 3) evicted data blocks that are later accessed more than once (+1 R).

Figure 3(c) shows that, on average, the vast majority of addresses that were evicted from a local bank were later reinserted once or more than once (over 60% and over 70% of reused addresses, respectively). Moreover, fewer than 60% of the addresses that were not accessed after eviction were evicted from a local bank. In general, this trend in which the percentage of the non-reinserted evicted addresses from a local bank is lower than that of the evicted and later inserted addresses is consistent for all workloads.

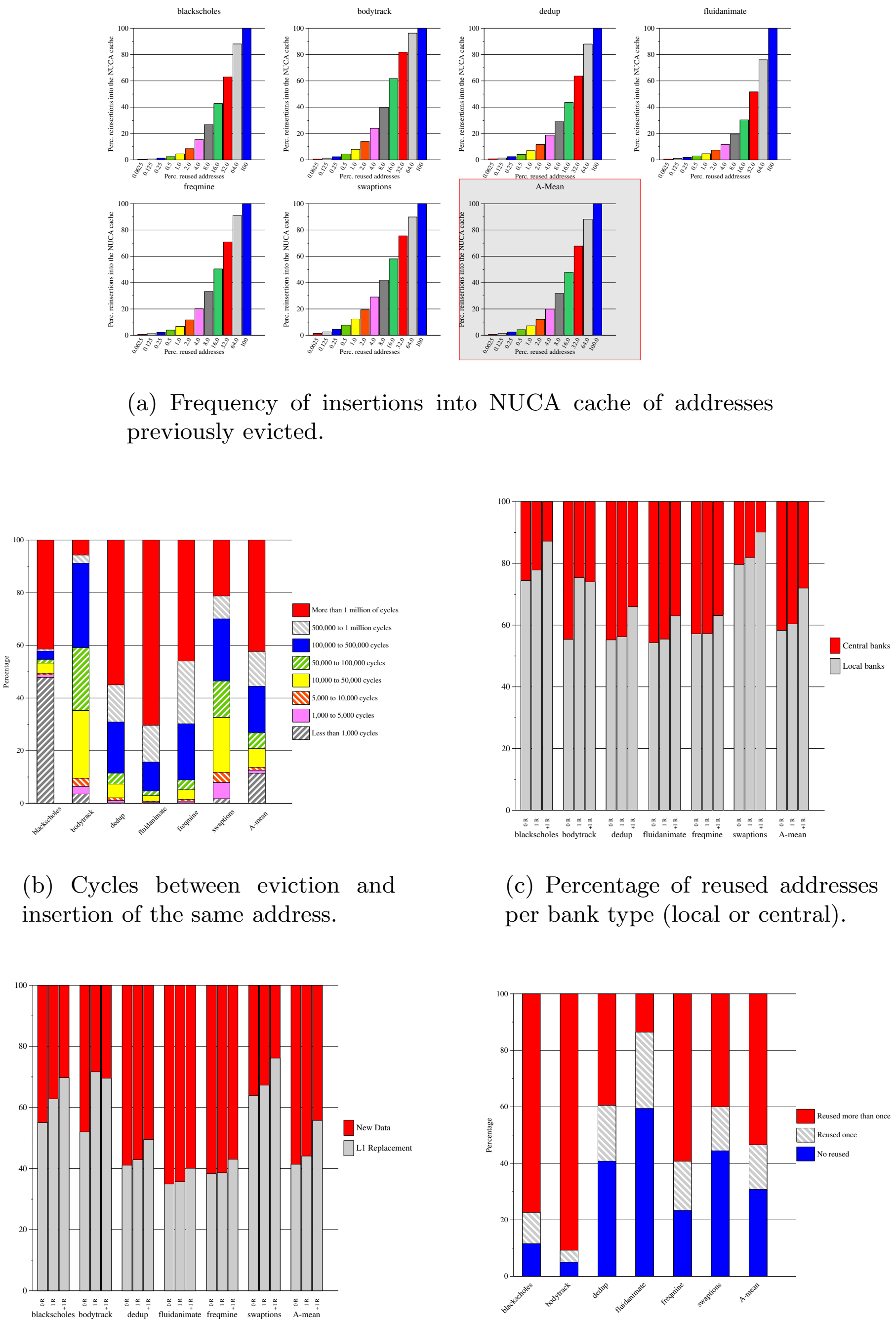
5.4 Action that provokes replacement

Finally, we analysed the type of action that provokes the eviction of the data block from the NUCA cache. Only two actions motivate an eviction from a NUCA cache bank: *incoming data from memory (New Data)* or *an eviction from L1 cache (L1 replacement).* When an incoming data block from off-chip memory is inserted into the NUCA cache, it is placed statically in any of the banks of the entire NUCA architecture. On the other hand, when a data block comes from an L1 cache eviction, it is always placed in the closest local bank to the L1 cache that provoked the replacement.

Figure 3(d) shows the percentage of evictions provoked by each of the actions, broken down into the following categories: addresses that were not accessed after their eviction; addresses that were inserted into the NUCA cache only once after their eviction; and addresses that were evicted and later inserted into the NUCA cache more than once. On average, 55% of the evictions of addresses that were reused more than once were evicted due to L1 replacement. Meanwhile, the same action provoked nearly 45% of the evictions of addresses that were reinserted only once into the NUCA cache. On average, only 40% of non-reused addresses were evicted due to an L1 cache eviction. In general, for all workloads the percentage of evictions provoked by L1 cache replacements that are not accessed later is always lower than in the other two cases.

6 Last Bank Optimisations

Section 4 shows that some performance benefits are achieved by adding Last Bank to the baseline NUCA cache architecture. However, it also shows



(a) Frequency of insertions into NUCA cache of addresses previously evicted.

(b) Cycles between eviction and insertion of the same address.

(c) Percentage of reused addresses per bank type (local or central).

(d) Action that provokes eviction.

(e) Percentage of reused addresses in the NUCA cache.

Fig. 3. Analysis of NUCA cache line replacements.

considerable potential improvements in performance that cannot be attained using the current Last Bank configuration. This section introduces two mechanisms that optimise the usage of Last Bank based on the analysis described in Section 5.

6.1 Selective Last Bank

Section 5.2 shows that almost 30% of evicted addresses that are later accessed return to the NUCA cache in a reduced number of cycles (fewer than 100,000 cycles). However, Last Bank cannot take advantage of this fast return and, with the exception of *blackscholes*, it does not result in any IPC improvement at all (see Figure 2). This is mainly because Last Bank is not large enough to deal with all the evicted data blocks from the entire NUCA cache. So, Last Bank is polluted with *useless* data blocks that will not be accessed again and that provoke the eviction of *useful* data blocks from Last Bank before they are accessed.

Based on these observations, we propose a selection mechanism in Last Bank called *Selective Last Bank*. This selection mechanism allows evicted data blocks to be inserted into Last Bank by way of a filter. The aim is to prevent Last Bank from becoming polluted with data blocks that are not going to be accessed further by the program.

Section 5.3 shows that almost 70% of reused addresses were evicted from local banks. On the other hand, only 60% of addresses that were not requested further after NUCA cache eviction were evicted from local banks and the rest were evicted from central banks. We therefore propose a filter that allows only the evicted data blocks that resided in a local bank before eviction to be cached.

Finally, the action that provokes the eviction from the NUCA cache could provide another filter for *Selective Last Bank*. However, after evaluating the filters for both incoming data from the off-chip and for L1 cache evictions, they were both ruled out because they complement one another and do not provide benefits to Last Bank.

6.2 LRU prioritising Last Bank

Figure 3(e) shows that the vast majority of evicted lines that return to the cache memory leave the NUCA cache and are later reused at least twice. Thus, we propose modifying the data eviction algorithm of the NUCA cache in order to prioritise the lines that enter the NUCA cache from Last Bank. We call this *LRU prioritising Last Bank (LRU-LB)*. LRU-LB gives the lines that have been stored by the Last Bank and that return to the NUCA cache an extra chance, so they remain in the on-chip cache memory longer. This requires storing an extra bit, called the *priority bit*, attached to each line in the NUCA cache.

The LRU-LB eviction policy works as follows. When an incoming line comes to the NUCA cache memory from Last Bank, its priority bit is set. Figure 4(a) shows how this policy works when a line with its priority bit set is in the LRU position. The line that currently occupies the LRU position, then, clears its priority bit and updates its position to the MRU, and thus, the other lines in the

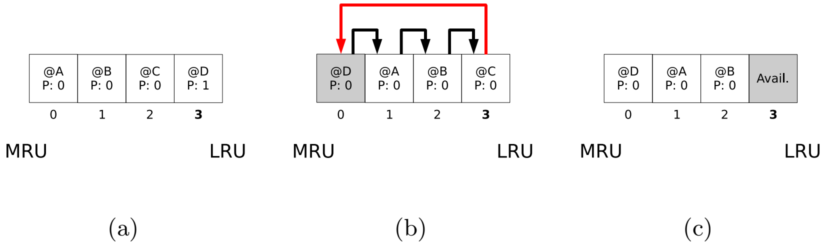


Fig. 4. LRU prioritising Last Bank (LRU-LB) scheme. (a) A priority line is in the LRU position. (b) The priority line resets its priority bit and updates its position to the MRU; the other lines move one position forward to the LRU. (c) The line in the LRU position is evicted since its priority is clear.

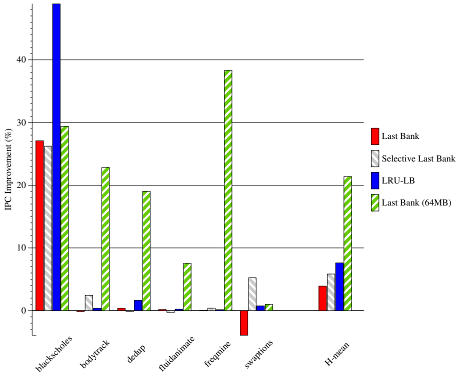


Fig. 5. Speed-up achieved with Last Bank optimisations.

LRU stack move one position forward towards the LRU (Figure 4(b)). Finally, as the line that is currently in the LRU position has its priority bit clear, it is evicted from the NUCA cache (Figure 4(c)). If the line that ends in the LRU position has its priority bit set, the algorithm described above is applied again until the line in the LRU position has its priority bit clear.

7 Results and Analysis

This section analyses the performance results obtained with the two optimisations for the Last Bank mechanism proposed in Section 6, *Selective Last Bank* and *LRU prioritising Last Bank (LRU-LB)*. With Selective Last Bank, the filter applied allows only the blocks that have been evicted from a local bank to be cached.

Figure 5 shows that, on average, both Last Bank optimisations achieve greater IPC improvement than that achieved by the Last Bank configuration.

Selective Last Bank obtains 6% speed-up with respect to the baseline configuration, while the performance gain obtained by LRU-LB is nearly 8%.

Figure 5 also shows that Selective Last Bank does not obtain performance benefits in three of the workloads (*blackscholes*, *dedup* and *fluidanimate*), however, it works especially well with *bodytrack* and *swaptions*. It is clear, then, that with an effective filter Selective Last Bank would provide significant performance benefits. Adaptive selective filters for Selective Last Bank will be analysed in future work.

With regard to the LRU-LB optimisation, giving an extra chance to reused addresses before being evicted from the NUCA cache has two direct consequences: 1) if accessed, they are closer to cores, and due to the NUCA basis they have lower access latency, and 2) the number of reused addresses stored in the NUCA cache is higher. As a result, LRU-LB outperforms the Last Bank configuration with all simulated workloads. We highlight that with the *blackscholes*, LRU-LB dramatically outperforms the Last Bank configuration (even the unaffordable setup) by achieving 49% IPC improvement.

8 Related Work

Kim et al. [13] introduced the concept of Non-Uniform Cache Architecture (NUCA). They observed that the increase in wire delays would make cache access times no longer a constant. Instead, latency would become a linear-function of the line's physical location within the cache. From this observation, several NUCA architectures were designed by partitioning the cache into multiple banks and using a switched network to connect these banks. The two main architectures, however, were Static NUCA (S-NUCA) and Dynamic NUCA (D-NUCA). Both designs organise 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 the development of these two architectures, several works using NUCA architectures have appeared in the literature. One of the most relevant proposals is NuRAPID [7], which decouples data and tag placement. NuRAPID stores tags in a bank close to the processor, optimising tag searches. Whereas NUCA searches tag and data in parallel, NuRAPID searches them sequentially. This increases overall access time but provides greater power efficiency. Another difference between NUCA and NuRAPID is that NuRAPID partitions the cache into fewer, larger and slower banks. In terms of performance, NuRAPID and D-NUCA achieve similar results, but NuRAPID vastly outperforms D-NUCA in power efficiency.

However, the introduction of CMP architectures posed additional challenges to the NUCA architecture leading Beckmann and Wood [3] to analyse NUCA for CMP. They demonstrated that block migration is less effective for CMP because 40-60% of the 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. [7] also proposed a version of NuRAPID for CMP in which each core had a private tag array instead of a single, shared tag array.

Recent studies have explored policies for bank placement [11], bank migration [12] and bank access [18] in NUCA caches. Kim et al. [13] proposed two alternatives for bank replacement policy: *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 element is moved to a lower-priority bank farther from the processor.

9 Conclusions

In this paper we propose adding an extra bank to deal with data blocks evicted from the NUCA cache. We call this bank *Last Bank*. This extra bank, which acts as the last-level cache on the chip, gives evicted data blocks a second chance to be reinserted into the NUCA cache without leaving the chip. We found that although this mechanism provides significant performance potential, the benefits achieved with Last Bank are strictly limited by the number of lines that can be allocated.

We therefore analysed the behaviour of the cache line replacements in the NUCA cache and propose two optimisations that provide additional IPC improvement to Last Bank without incurring unaffordable implementation costs.

We first propose *Selective Last Bank*, which defines that only evicted data blocks that pass the selection filter are allowed to be allocated in Last Bank. In this paper we propose a selection filter that allows only the data blocks that were evicted from a local bank to be cached. With Selective Last Bank we achieve up to 6% IPC improvement with respect to the baseline configuration, however, the performance results obtained by this mechanism rely on the effectiveness of the filter applied. In future works we will analyse adaptive selection filters.

Finally, we propose *LRU prioritising Last Bank*. This modifies the data eviction algorithm of the NUCA cache by prioritising the data blocks that came from Last Bank. On average this mechanism achieved performance gains of up to 8% with respect to the baseline configuration.

10 Acknowledgements

This work is supported by the Spanish Ministry of Science and Innovation (MCI) and FEDER funds of the EU under contract TIN 2007-61763, 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. V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger. Clock rate vs. ipc: The end of the road for conventional microprocessors. In *27th International Symposium on Computer Architecture*, 2000.
2. B. M. Beckmann, M. R. Marty, and D. A. Wood. Asr: Adaptive selective replication for cmp caches. In *In 39th Annual IEEE/ACM International Symposium of Microarchitecture (MICRO-39)*, 2006.
3. B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *37th International Symposium on Microarchitecture (MICRO-37)*, 2004.
4. C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *Procs. of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.
5. J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *Procs. of the 21st ACM International Conference on Supercomputing (ICS-21)*, 2007.
6. J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *Procs. of the 33rd International Symposium on Computer Architecture*, ISCA '06.
7. 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 (MICRO-36)*, 2003.
8. H. Dybdahl and P. Stenström. An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors. In *IEEE 13th International Symposium on High-Performance Computer Architecture*, 2007.
9. P. Frost Gorder. Multicore processors for science and engineering. *Computing in Science & Engineering*, March-April 2007.
10. Z. Guz, I. Keidar, A. Kolodny, and U. C. Weiser. Nahalal: Cache organization for chip multiprocessors. *IEEE Computer Architecture Letters*, 2007.
11. 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 (ICS-19)*, 2005.
12. M. Kandemir, F. Li, M. J. Irwin, and S. W. Son. A novel migration-based nuca design for chip multiprocessors. In *ACM/IEEE conference on Supercomputing*, 2008.
13. C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 2002.
14. R. Low. Microprocessor trends: multicore, memory, and power developments. *Embedded Computing Design*, September 2005.
15. 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*, volume 35-2, pages 50–58. Computer, 2002.
16. 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*, September 2005.
17. D. Matzke. Will physical scalability sabotage performance gains? *IEEE Computer*, September 1997.

18. N. Muralimanohar and R. Balasubramonian. Interconnect design considerations for large nuca caches. In *Procs. of the 34th International Symposium on Computer Architecture*, ISCA'07.
19. N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40)*, 2007.
20. M. M. Zahran. Acm sigarch computer architecture news. *ACM SIGARCH Computer Architecture News*, March 2003.
21. M. Zhang and K. Asanović. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *Procs. of the 32nd International Symposium on Computer Architecture*, ISCA '05.