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

Javier Lira
Universitat Politècnica de Catalunya
javier.lira@ac.upc.edu

Carlos Molina
Universitat Rovira i Virgili
carlos.molina@urv.net

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

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 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.

1. 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 [23], 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 [12, 19]. There are two basic reasons why the major processor manufacturers devote so much of their research to this type of design. First, the increasing need to provide better performance means that energy consumption starts to cause impractical heat dissipation. Therefore, designing a processor with simple cores reduces consumption while maintaining performance. Second, the existing gap between processor speeds and memory access is getting wider. During the last decade this has led to an approach involving concurrent execution, initially through the execution of multiple threads in one processor and now with the inclusion of multiple cores in a single chip.

For these reasons and because of the steady increase in the number of transistors that can be integrated on a chip, 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 [1].

A couple of challenges have arisen from the fact that multicore systems will dominate the market by the next decade: (a) the bandwidth and the complexity of network link connecting to different cores, and (b) sharing and consistency of data in the memory hierarchy of multicore processors. 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 [7, 8, 14], victim replication [26], adaptive selective replication [3] and other works that exploit the private/shared cache partitioning scheme [11, 13].

Current multicore systems incorporate larger and shared second-level caches with a homogeneous access time. Cache size is expected to grow as bandwidth requirements increase and processing technology shrinks. Therefore, overall wire delays will make it unfeasible to maintain a constant latency across the cache [2, 22]. Non-Uniform Cache Architectures (NUCA) have been proposed [18] to deal with the growing memory latencies in large on-chip caches.

This paper aims to analyze how Non-Uniform Cache Architecture performs on a Chip Multiprocessor using the Parsec 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.

MMCS'09, March 7, 2009, Washington DC, USA

The remainder of this paper is structured as follows. Section 2 identifies the underlying concept behind a NUCA system and the policies that characterize it. Section 3 presents the baseline model that has been assumed, the simulation tools and a brief description of the benchmarks used. Section 4 describes the alternatives of each bank policy considered in this study. Section 5 presents the results obtained during the simulations. Related work is summarized in Section 6. Finally, Section 7 outlines the main conclusions of this work.

2. NUCA

Traditional cache architectures assume that each level in the cache hierarchy has a single and uniform cache access time. 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. However, this non-uniformity may be exploited to provide master access to cache lines in those portions of the cache that are closer to the processor. This technique is known as Non-Uniform Cache Architecture (NUCA), and was first proposed by Kim et al. [18].

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:** 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.
- **Bank Access Policy:** This policy determines the bank-searching algorithm in the NUCA cache memory space. Other considerations are also determined by this policy, such as whether all banks should be accessed in parallel, sequentially or progressively, depending on the latency of banks.
- **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.
- **Bank Replacement Policy:** This policy determines how NUCA architecture behaves when there is a data eviction from one of the banks. For instance, when a data element is evicted from a bank it can be sent directly to the off-chip memory, or it can be located in another bank.

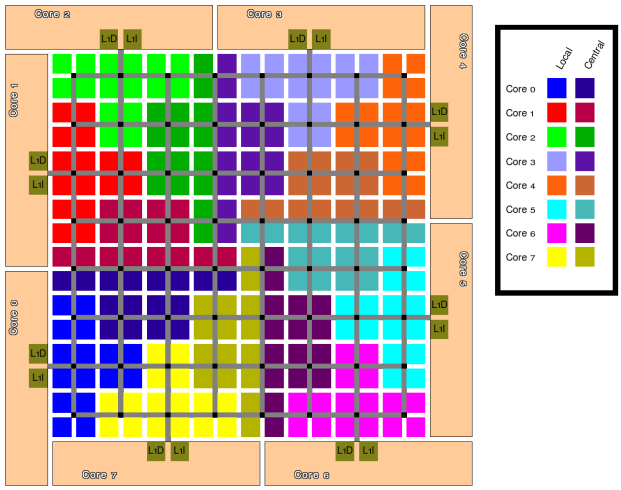


Figure 1: Organization of NUCA architecture.

3. EXPERIMENTAL FRAMEWORK

This section describes the experimental framework (the baseline model and simulation tools) assumed in this work.

3.1 Baseline Model

This paper deals with an L2 NUCA organization based on that proposed by Beckmann and Wood [4]. Figure 1 illustrates this baseline model. A die with 8 cores on the edges and a NUCA shared second-level cache in the center has been used. Each core maintains its own private first level cache that is divided for data and instructions. All caches on both levels are 4-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 cache with the third level of the 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 1 summarizes the CMP parameters assumed for the baseline model.

3.2 Simulation Tools

We used Simics [20], a full system execution-driven simulator extended with the GEMS (General Execution-driven Multiprocessor Simulator) toolset [21], 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

Number of cores	8
Core processor	Out-of-order SPARCv9
Main memory size	4 GBytes
Memory bandwidth	512 bytes/cycle
On-chip wire delay	1 cycle
Off-chip wire delay	20 cycles
Switch delay	1 cycle
Private L1 data caches	8 KBytes
Private L1 instruction caches	8 KBytes
Shared L2 NUCA cache	1 MB, 256 banks

Table 1: CMP parametrization.

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 [25] greatly assists circuit simulation. It estimates the area, access and cycle 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.

3.3 PARSEC Benchmark Suite

The Princeton Application Repository for Shared-Memory Computers (PARSEC) has been recently released [6]. This benchmark suite includes emerging applications and commercial programs that cover a wide area of working sets and allow current Chip Multiprocessor technologies to be studied more effectively [5].

In this paper we evaluate a subset of the PARSEC benchmark suite. We also consider the simlarge inputs of PARSEC benchmarks in our simulations. Furthermore, we put forward a significant number of instructions for preventing initialization behavior and thread creation of programs. We then fill the cache by executing 100 million instructions and finally we collect the statistics for the following 500 million instructions.

4. BANK POLICY APPROACHES

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

4.1 Bank Placement Policies

This policy determines where data can be located during its life in the L2 NUCA cache and where it is first placed when it comes from the off-chip memory of L1 caches. 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:** Here, there is just one bank. The data is always placed in the same bank and this depends on the address bits of the data block.

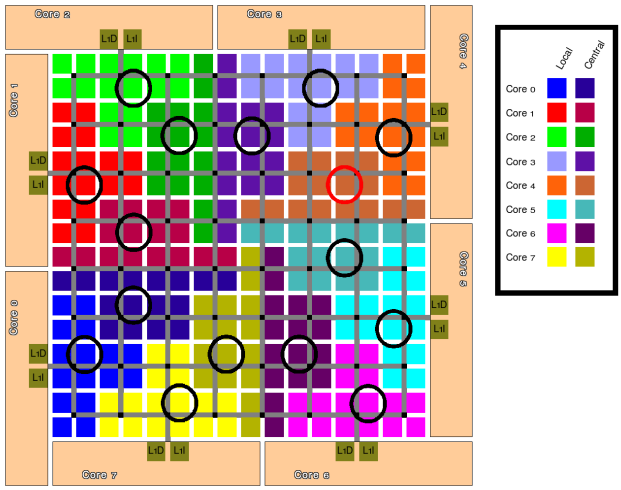


Figure 2: 16B + Static Bank Placement Policy.

- **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.

As an example, Figure 2 shows the 16 possible placement positions of a data block into the NUCA cache using the *16B + Static* Bank Placement policy. This policy also determines that the initial position of data blocks within the NUCA cache is statically defined. This bank is marked with a red circle in the figure.

4.2 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, parallel search or a combination of both. Notice that a serial search will reduce the number of bank lookups but will increase the miss resolution time. On the other hand, a parallel search reduces miss resolution time but increases the number of bank lookups and thus the energy used. We evaluate the following approaches:

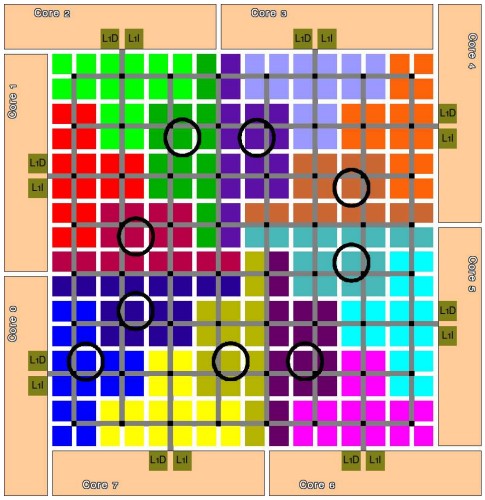
- **Partially Serial:** This approach combines the serial and parallel searches but giving more weight to serial access. Assuming for instance that data can be located in 16 banks (one local and one central of each core), the

searching algorithm is divided in two steps. The first step sequentially searches for data in banks (1 local and 8 central in total), starting with the local bank of the requesting core and finishing with the bank furthest away among the central ones. If the data block is not found, the second step is triggered. In this case, the 7 remaining banks (local to other cores) are accessed simultaneously.

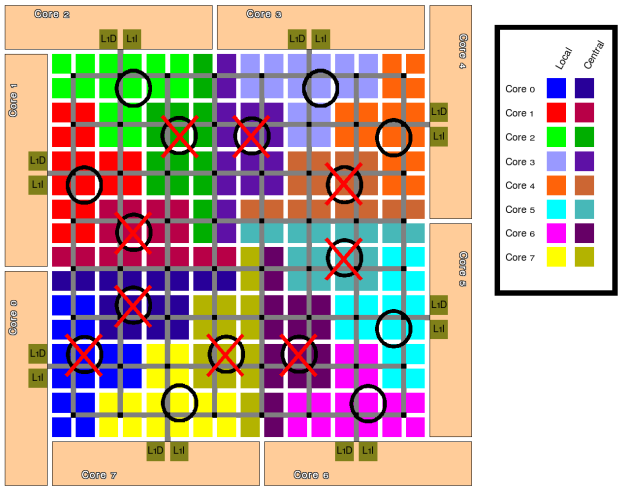
- **9P + 7P:** The searching algorithm is also divided in two steps, giving more weight to parallel access. The first step searches in parallel the local bank of the requesting core and all the central ones (9 banks in total). If the data block is not found, the second step is triggered to simultaneously search the 7 remaining banks (local to other cores).
- **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.

For the sake of clarity, Figure 3 shows an example of how to search the requested data block in the NUCA cache using the *9P + 7P* Bank Access Policy. The example of the figure assumes that core 0 is accessing a data block. Thus, the searching algorithm works as follows:

- **Step 1:** The local bank of core 0 and all the central banks where the data block can be placed are accessed in parallel. Then, if after all accessed banks have responded, the data block is still missing, the second step of the data search algorithm is launched.
- **Step 2:** The other NUCA cache banks where data can be placed are accessed, also in parallel, i.e. the local banks of all cores that differ from the requestor core.
- **Extra step:** If the requested data block is still not found, the data access request is sent to the next level of the memory hierarchy (in this case, the main memory).



(a) First step: Access to local bank and 8 central banks.



(b) Second step: Access to the other local banks.

Figure 3: 9P + 7P Bank Access policy.

4.3 Bank Migration Policy

This policy allows data movements among banks in order to reorganize data blocks in the NUCA cache. 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.
- **Gradual + 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.
- **Gradual + 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.

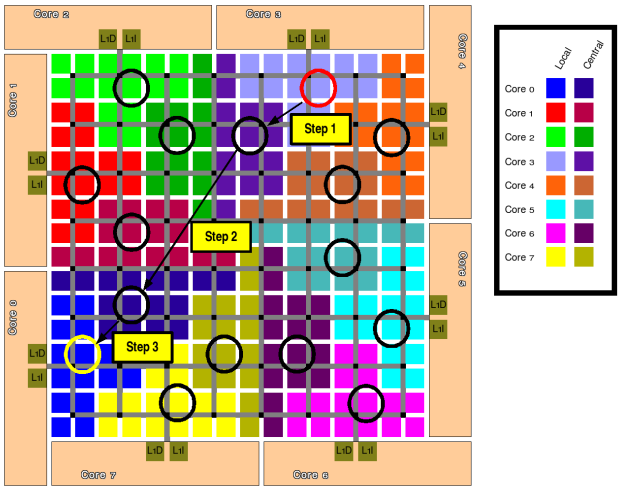


Figure 4: Gradual + Swapping Bank Migration policy.

In order to clarify how the *Gradual + Swapping* Bank Migration policy works, Figure 4 shows the data block movements described within the baseline NUCA cache architecture by assuming the following situation:

- **Step 1:** Core 0 accesses a data block that is placed on a local bank of core 3, so this data block is promoted to the central bank of core 3.
- **Step 2:** Core 0 accesses the same data. The data block is located in the central bank of core 3 and is therefore migrated to the central bank of core 0.
- **Step 3:** Core 0 accesses the same data block for a third time, so this data block, which is currently placed in the central bank of core 0, is moved into the local bank of core 0, which is the closest bank where it can be placed.
- **Extra step:** Core 0 accesses the same data again but no migration is applied because it is already in the local bank of core 0.

4.4 Bank Replacement Policy

The insertion of an incoming data block into a bank may cause the eviction of another data block. 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 [16] can be added to store all the evicted data from the NUCA cache. In this case, this bank behaves as a victim cache [17] instead of sending data to the off-chip memory when there is a replacement. Furthermore, when a hit occurs on the *Last Bank*, the data is moved to the regular NUCA cache.

5. 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: **Parallel**
- Bank Migration Policy: **Gradual + Swapping**
- Bank Replacement Policy: **Zero Copy**

5.1 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 cache. *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.

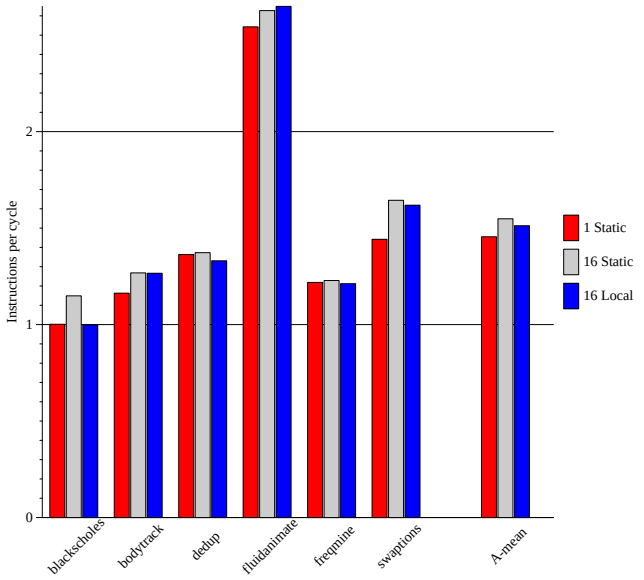


Figure 5: Instructions per cycle of bank placement policy alternatives.

Figure 5 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.

5.2 Bank Access Policy

In this policy we also analyze two opposite approaches (partially serial and parallel) as well as a more affordable approach that better combines serial and parallel access. *Partially serial* access saves energy but provides higher access time. A *parallel* search in bank access policy provides the lowest access time but seriously increases the energy used since all banks must be accessed. The trade off approach searches in two steps, each of them are parallel but one after the other.

Figure 6 shows the huge performance potential of complete parallel access. It heavily outperforms the other two approaches. In particular, *parallel* access achieves an average IPC of 1.5, whereas *partially serial* access is limited to 1.05, and *9P+7P* achieves an average IPC close to 1.2. *9P+7P* is better than *partially serial* access but is still a long way from complete *parallel* access. These results suggest the broad area of improvement that this policy introduces. Therefore, greater efforts must be made to find new alternatives to current bank access policies.

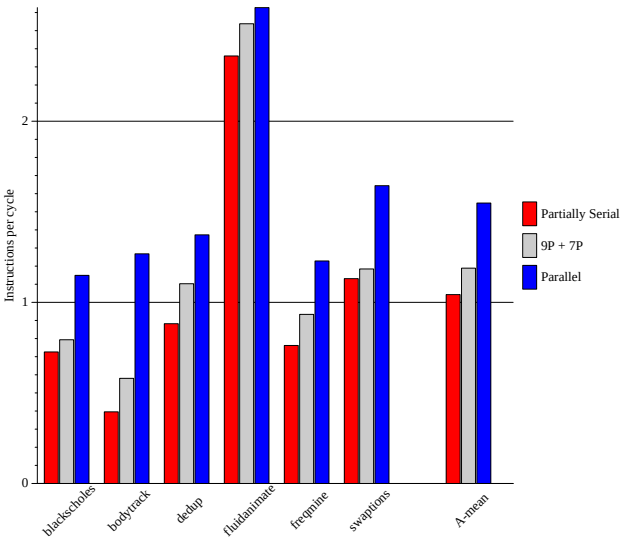


Figure 6: Instructions per cycle of bank access policy alternatives.

5.3 Bank Migration Policy

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

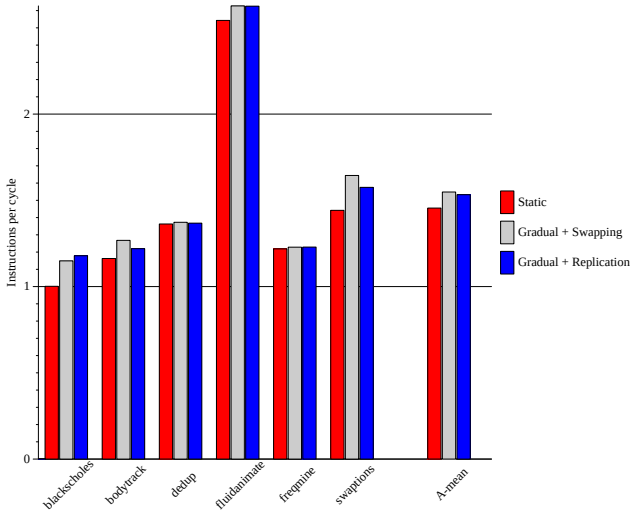


Figure 7: Instructions per cycle of bank migration policy alternatives.

Figure 7 shows the performance potential of a migration policy. We observe that migration performs well in all cases. Therefore, a non-migration approach is close to the migration approaches. As outlined in Section 5.1, we believe that there is a high correlation between the bank placement and bank migration policies. Therefore, further analysis will need to combine both policies.

5.4 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 that has been located on the chip at the same distance from all cores which is the last level cache between the NUCA cache and the off-chip memory. Regarding the access policy, this last bank is included in the second step of the baseline configuration. Thus, a parallel access of 7 central banks and the last bank is assumed. We also assume a huge non-affordable *Last Bank* of 16 MBytes in order to estimate the potential benefit of this mechanism. Figure 8 shows the instructions-per-cycle results for each alternative.

Generally, *One Copy* outperforms the *Zero Copy* and *Last Bank* configurations. On average, *One Copy* achieves 1.25 of IPC whereas *Zero Copy* and *Last Bank* achieve 1.20 and 1.22, of IPC, respectively. The figure also shows promising performance potential when the unbounded *Last Bank* configuration is assumed. Therefore, a better exploration of the last bank approach is needed.

6. RELATED WORK

Kim et al. [18] introduced the concept of Non-Uniform Cache Architecture (NUCA). They observed that increasing wire delays would mean that cache access times were no longer constant. Instead, latency would become a linear-

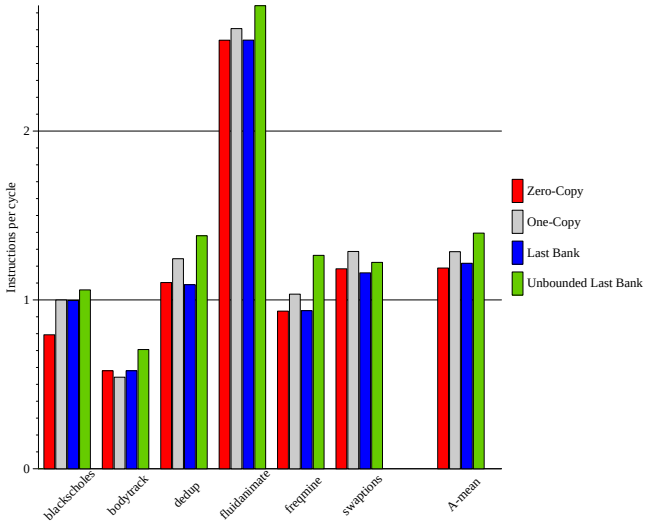


Figure 8: Instructions per cycle of bank replacement policy alternatives.

function of the line's physical location within the cache. On the basis of this observation, they designed several NUCA architectures by partitioning the cache into multiple banks and using a switched network to connect these banks. Two main alternatives have been proposed: Static NUCA (S-NUCA) and Dynamic NUCA (D-NUCA). Both designs organize the multiple banks into a two-dimensional switched network. The difference between the two architectures is the *Placement Policy* they manage. While in S-NUCA architecture, data are statically placed in one of the banks and always in the same bank, in D-NUCA architecture data can be promoted to be placed in closer and master banks. Although this promotion allows 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. Two alternative bank replacement policy are proposed: *zero-copy* and *one-copy*.

CMPs present additional challenges for on-chip cache management. First, a cache on a CMP requires multiple ports to provide appropriate bandwidth. Second, multiple threads mean multiple working sets, which compete for limited on-chip storage. Finally, sharing code and data interfere with block migration, since one processor's low-latency bank is other processor's high latency bank. Beckmann and Wood [4] gathered the current proposals for managing wire delays and combined them with Chip Multiprocessors. They demonstrated that block migration is less effective for CMP because 40-60% of hits in commercial workloads were satisfied in the central banks. Block migration effectively reduced wire delays in uniprocessor caches. However, to improve CMP performance, the capability of block migration relied on a smart search mechanism that was difficult to implement.

Huh et al. [15] introduced 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. An ideal design would capture the benefits of both reduced misses and reduced hit latencies. 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. They also concluded that the simplest design– an S-NUCA organization with a small sharing degree–was probably the best.

Muralimanohar and Balasubramonian [24] 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.

Lira [16] has provided a detailed analysis of bank replacement policy. They analyze several replacement alternatives and study the characterization of those data that are evicted and then reinserted in the future. Furthermore, they propose 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 propose filtering evicted data so that the *Last Bank* only contains those data blocks, which are most likely to be accessed in the near future.

Chishti et al. [9] proposed an alternative to NUCA architecture named Non-uniform access with Replacement And Placement usIng Distance associativity (NuRAPID). This architecture is based on decoupling data and tag placement. NuRAPID stores tags in a bank close to the processor, optimizing tag searches. Whereas NUCA searches tag and data in parallel, NuRAPID and D-NUCA achieve similar results, although NuRAPID heavily outperforms D-NUCA in power efficiency. The NuRAPID version for CMP is known as CMP-NuRAPID and was also proposed by Chishti et al. [10]. This proposal mitigates some of the effects described by Beckmann and Wood [4].

7. CONCLUSIONS AND FUTURE WORK

This paper analyzes how NUCA organizations perform according to different approaches, each using the four policies that characterize their behavior (placement, access, migration and replacement). The Parsec 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. A more affordable mechanism combining parallel and serial searches is a good trade off, but this is still far from ideal.

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. Anyway, results do not 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.

8. 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.

9. REFERENCES

[1] http://view.eecs.berkeley.edu/wiki/chip_multi_processor_watch.

[2] 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.

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

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

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

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. In *International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[7] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *33rd International Symposium on Computer Architecture*, 2006.

[8] J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *21st ACM International Conference on Supercomputing*, 2007.

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

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

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

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

[13] Z. Guz, I. Keidar, A. Kolodny, and U. C. Weiser. Nahalal: Cache organization for chip multiprocessors. *IEEE Computer Architecture Letters*, 2007.

[14] E. Herrero, J. Gonzalez, and R. Canal. Distributed cooperative caching. In *17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.

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

[16] J.Lira. Data replacement policy on non-uniform cache architecture for chip-multiprocessors. Master's thesis, Universitat Politécnica de Catalunya, 2008.

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

[18] 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*, 2002.

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

[20] 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.

[21] 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.

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

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

[24] N. Muralimanohar and R. Balasubramonian. Interconnect design considerations for large nuca caches. In *34th International Symposium on Computer Architecture*, 2007.

[25] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Cacti 6.0: A tool to understand large caches. Technical report, University of Utah and Hewlett Packard Laboratories, 2007.

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