Exploring the relationship between architectures and management policies in the design of NUCA-based chip multicore systems

Exploring the Relationship between Architectures and Management Policies in the design of NUCA-based Chip Multicore Systems

Sandro Bartolini[ϕ], Pierfrancesco Foglia[ψ1], Cosimo Antonio Prete[ψ]
[ψ]*Dipartimento di Ingegneria dell'Informazione, Università di Pisa*
Via Diotisalvi, 2 – 56100 Pisa (PI) Italy
{foglia, prete}@iet.unipi.it
[ϕ]*Dipartimento di Ingegneria dell'Informazione e Scienze Matematiche*
Università degli Studi di Siena, via Roma 56 – 53100 Siena (SI), Italy
bartolini@dii.unisi.it

Abstract

The last level on-chip cache (LLC) is becoming bigger and more complex to effectively support the various localities coming from multiple cores and threads running concurrently in modern processors. Furthermore, LLC design can be constrained by various restrictions that limit the freedom in their organization, for example in the relative positioning and clustering of processing cores and cache banks. Non Uniform Cache Architectures (NUCAs) offer a hierarchy of access times, which can be usefully exploited by the NUCA management policies (i.e. the ways in which data are either mapped to cache banks and/or moved among them upon access) to achieve high performance and low power consumption. The objective of the work is to single out the optimal combination of data management policies and cache-core layouts and to highlight which is the most performing one. With this aim, we compare two basic layouts for NUCA based systems, the first with cores connected to only one side of the shared NUCA cache (one-side), the second with half of the cores on one side and the others at the opposite side of the NUCA (two-sides). For all the configurations, we evaluate the effectiveness of both static and dynamic NUCAs and, where applicable, we consider also optimizations based on profile-guided bank remapping and replication of shared copies. As overall design guidelines, our results show that the one-side layout achieves the best performance and the lowest power consumption with the considered hw-sw optimizations. Then, similar results can be achieved in the two-sides layout only by introducing more sophisticated copy replications. Lastly, software based profile driven optimization allows the system to achieve the lowest usage of network resources.

Keyword: multicore architecture, sub-banked last level cache structure, layout, design space exploration, data replication, profile-based memory layout optimizations.

[1]Corresponding author

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

1 Introduction

The design of an efficient memory hierarchy for Chip MultiProcessor (CMP) [1] systems is currently even more critical than in the past. In fact, multiple on-chip cores [1] induce an aggregated bandwidth request towards the memory hierarchy which is becoming a critical design issue. Even if main memory bandwidth, and especially latency is slowly improving, it still remains far below the requirements and can constitute a major performance bottleneck. This is particularly critical in server workloads like DBMS, business data analytics and scale-up High Performance applications, that represent the main target of the architectures considered in this work [57][75][76]. For these reasons, on-chip cache hierarchy, through its capacity and latency, has a strategic role in translating computational potential of CMPs into overall system performance.

NUCA (Non Uniform Cache Architecture) approach used as a Last Level Cache (LLC) design paradigm [5] is promising in this scenario because of its capability to tolerate wire delays [4], and its intrinsic scalability towards both suitable sizes and high number of access ports [52]. However, NUCA caches, being based on a banked organization and on an interconnection network between banks, introduce additional design space dimensions [11][12][52][56][72] to be explored and specific issues to be addressed to seek optimal performance.

A NUCA cache is characterized by an *architecture* and by *data management policies*. The architecture is defined by the way in which banks are connected among them and by their logical placement with respect to the cores (layout). Data management policies are defined by the mapping of memory addresses to cache banks, and by the specific mechanisms adopted to exploit the non-uniformity of access time, which in a NUCA cache depends on the physical location of the bank to be accessed. According to the mapping, a NUCA cache can be classified as Static (SNUCA), where each memory address is mapped to a specific cache bank (and line) and Dynamic (DNUCA) where, a) each bank is mapped to a set of banks that introduce a hierarchy of access times, and b) a migration policy is defined to move the line inside the set of banks [5]. Data management policies can have a big impact on performance/power consumption figures. For instance, migration can be effective in reducing access latency by moving mostly used data near to the requesting cores at runtime. Similarly, a feedback-driven mapping function of memory lines to cache banks can achieve a similar objective operating at compile time to reduce the average hit-latency of the cache. Also the layout may have a big impact on the performance of a system adopting a NUCA cache, as it changes the distance among cores and NUCA banks. Furthermore the layout may interact also with the management policies; for instance, the migration mechanism is less effective when the requesting cores are placed at the opposite sides of the cache due to the “ping-pong” or “conflict hit” effects [2], [6], [40].

The layout may be constrained by external factors such as wire placing, connectivity towards the outside of the core or to other on-chip elements [67] (i.e. NVM, interconnection networks, memory controllers, etc), and the need to manage thermal hotspots [68][69]. At the same time, to dominate the complexity of floor-planning in processors that have billions of transistors, methodologies exploiting modules with regular designs are more and more widely adopted [70]. Such possible constraints may indicate that one layout should be preferable given some specific contour conditions in the design. However, to the best of our knowledge, it is not clear which is the best layout (in terms of performance or power saving) when varying the NUCA architecture and management policies or, from the reverse standpoint, which combination of data management policies allows the achievement of the best performance/power figures for a given layout.

The objective of this work is to highlight the relationships between NUCA management policies and layouts, and to propose guidelines that allow system designers to identify the best performing and power efficient solutions, also considering the possible constraints in the overall design.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

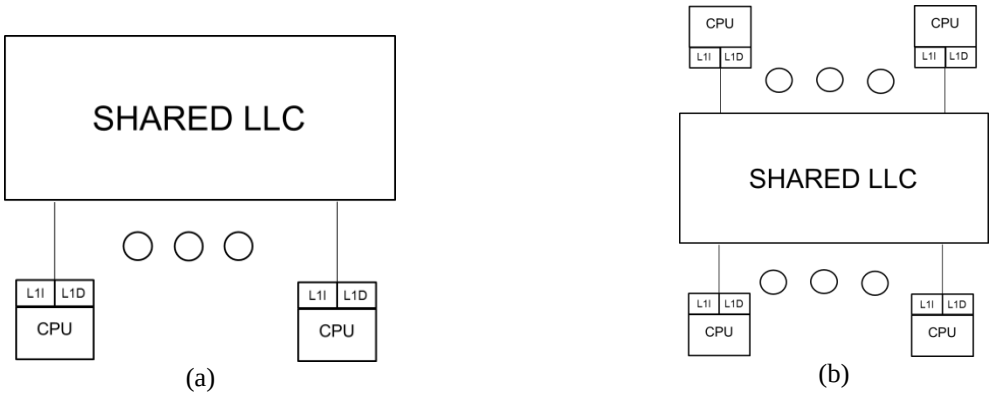


Figure 1: (a) one-side layout and (b) two-sides layout of a chip-multiprocessor system.

There are two basic layouts adopted in the design a CMP system (Figure 1): "one-side" and "two-sides". In the first one all cores are placed on the same side of the shared LLC. In the second one half cores are positioned on one side and the other half on the other. The scheme "one-side", for instance, is adopted in the Intel Core Gulftown microarchitecture (e.g. Core i7 980X - 6 cores) [20], in the Intel Core Sandy Bridge [21] or Ivy Bridge microarchitecture (e.g. i7 2600K or i7-3770K - 4 cores+ GPU)), in the Xeon Broadwell microarchitecture [74] and in the AMD Opteron Magny Cours [22]; the scheme "two-sides" is exemplified by the Intel core Sandy Bridge-E (i7 3960 - 6 cores)[19] and Ivy Bridge-EP microarchitecture, IBM Power 7 and Power 8 multicore families [23], [24], [25], Oracle SPARC M7 Processor [57]and has been proposed/evaluated in different research works[6], [26], [27]. Nowadays many commercial CMPs also exhibit a banked organization for the last level cache, that can be UCA (i.e. with Uniform Cache Access time) or NUCA depending on the behavior of the underlying interconnections. For instance, Power 7 and 8 CMP families utilize the banked L3 cache as a victim for the L2 cache, putting evicted L2 data in banks "near" to the L2 cache originating the data [23], [24], similarly to the Sparc M7 processor [57], de facto all leveraging NUCA features. Furthermore, some i7 processor sub-families [32] expose NUCA organizations for their L3 cache. However, a thorough comparison between the performance and power implications of the two NUCA layouts is still missing, as well as the identification of the best combination of NUCA management policies and NUCA architectures that achieves the best performance/power figures in each layout.

In this work, after reviewing the basic concepts of SNUCA and DNUCA cache organizations and their implementation issues, we analyze the potential impact on performance of the two layouts of Figure 1, and how each of them can be exploited within a NUCA cache. The considered specific layouts are shown in Figure 2, where we label the "one-side" one as *8P* and the "two-sides" one as *44P*. We evaluated both SNUCA and DNUCA architectures, along with a DNUCA optimized version, to solve problems arising in the two-sides layout, as well as a SNUCA scheme in which the mapping of memory blocks to cache banks is optimized to reduce cache hit-access time.

Our results allow the singling out the following guidelines. The 8P layout achieves the best performance and the lowest power consumption when using hw-sw optimizations. The 44P layout can achieve similar figures by adopting a more sophisticated scheme for the replication of copies in a DNUCA organization, while in this case software optimizations are less effective in identifying a mapping that significantly reduces the access time. Then, within SNUCA layouts, the considered software based optimization achieves the lowest network resources usage, because heavily accessed cache regions are mapped closer to the requesting cores.

The rest of this paper is organized as follows: Section 2 introduces basic concepts of the SNUCA and DNUCA schema. Section 3 discusses the combined impact of the layouts in Figure 2 and NUCA schemas on performance, while Section 4 presents solutions to minimize it. Section 5 presents our evaluation methodology and Section 6 discusses the achieved results and draws the overall design guidelines. Section 7 discusses related works and finally Section 8 concludes the paper.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

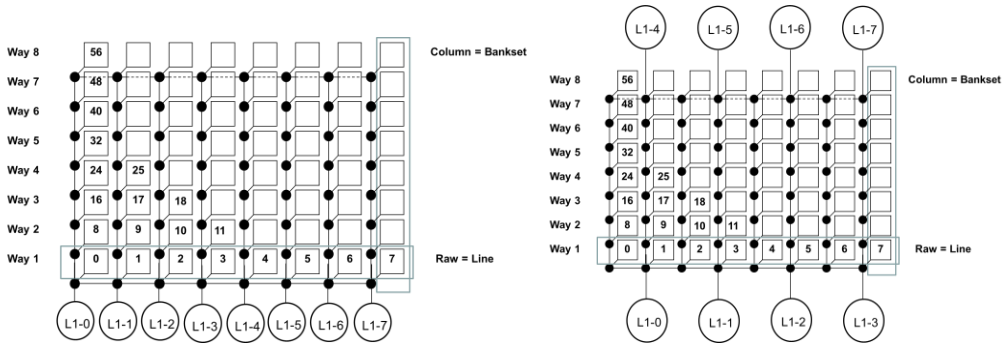


Figure 2: The two considered *layouts*: 8P and 44P. Squares are NUCA banks, black circles are NoC switches, white circles represent cpu+L1 caches. In the case of a DNUCA, bank columns correspond to banksets. Rows of the bank matrix are called *lines*, and are numbered from 1 (at the bottom) to 8.

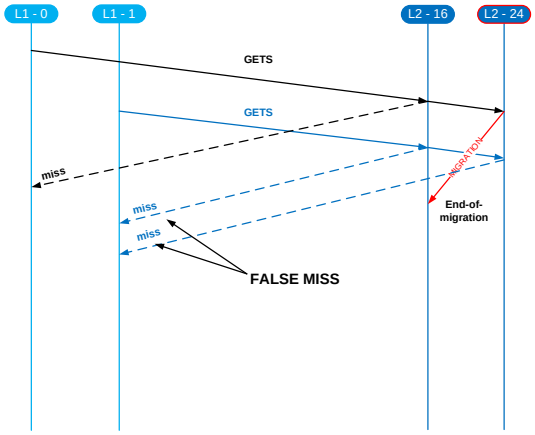


Figure 3: The false miss race condition. L1-0, with reference to the architectures of Figure 2, generates a request that produces a migration (a data migrates from L2-24 to L2-16) and a simple migration message (MIGRATION) is used. If another cache (L1-1) generates a new request for the same data, and such request arrives at L2-16 before the end of migration, both L2-16 and L2-24 caches reply with a miss, so for L1-1 the data is not present in the cache (FALSE MISS).

2 Basic NUCA design

NUCA caches are based on a multi-banked architecture (Figure 2), where each bank can be accessed independently from the other, and banks are connected via a switched network [7]. The access time to each bank depends on the physical distance of the cache bank from the requesting core. As such distance varies among banks, it leads to a Non-Uniform access time Cache Architecture.

A NUCA can be a Static NUCA (SNUCA), when each memory block is mapped (i.e. a copy can stay) only in one cache bank, or a Dynamic NUCA (DNUCA), when a memory block is mapped to a set of cache banks (called *bankset*) having different access time and the copy of a memory block may migrate among the banks belonging to the same bankset. Such flexible mapping is exploited to bring the most frequently used copies near the utilizing core, thus minimizing the access latency to such data. For this solution, a migration and a search mechanism are required. The migration mechanism must decide when a migration is triggered, and where the data have to be placed after migration. The usual adopted mechanism is a "1hit-1hop" or gradual promotion [5], i.e. on a hit the data moves one hop towards the requestor. The search mechanism is needed because data are not in a fixed location, because of migration. Due to its performance [26] and simplification in the required underlying support [51], [60], the search

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

mechanism considered as a baseline is the parallel search [5], that broadcasts messages in the bankset, i.e. a request for a data is sent to all the banks that may contain the data[2].

When implementing the migration of data in the last level DNUCA cache, some race conditions must be solved, otherwise there could be situations in which: 1) copies of the same data, with different values, may be present in cache, or 2) update operations of a lower level cache may produce error conditions [37].

A consequence of the search and migration mechanisms on a cached copy is the *false miss* problem (Figure 3, with reference to the systems of Figure 2). Consider two processes P0 and P1, running on two different cores, and sharing a variable A located in a block in the last level cache. If P0 accesses A, and, as a consequence of a miss, L1 asks the LLC for the copy, the copy in the LCC moves towards the core executing P0. When the copy is migrating, there is a time interval in which none of the banks of the bankset is able to provide to another requesting L1 cache the referred block. During such an interval, if P1 accesses A[3], it misses in its L1 cache and the request to the L2 cache results in a miss, even if the copy is actually in L2 (false miss). The effect of such a false miss is that a new copy of the data is taken from the memory, and, if the copy in cache has been previously modified, the two copies are now different. Figure 3 shows the false miss problem with a sequence diagram: the block migrates from L2-24 to L2-16 due to a request coming from L1-0, but the subsequent request coming from L1-1 is received by L2-24 *after* the block has migrated, and by L2-16 *before* the block is received, thus resulting in a miss and in a request to the external memory.

The *false miss* problem was first presented by Beckmann and Wood [2] and to face the *false miss* race condition, one solution consists in adding a block migration mechanism, called *False Miss Avoidance* (*FMA)* protocol [11] to the baseline protocol. The FMA protocol is able to prevent false misses by guaranteeing that, at any time, if a copy exists in the L2 cache, at least *one* bank knows where it is and so it is able to signal the hit condition, and forward the request to the bank holding the copy. In this way, even if a request arrives when the copy is *on-the-fly*, it won't result in a miss. The FMA protocol doesn't deny the other LLC-1 caches to issue new requests for a migrating block, as described in [6], nor relies on any centralized structure and *lazy migration* as in [2]. Figure 4 shows the main FMA actions.

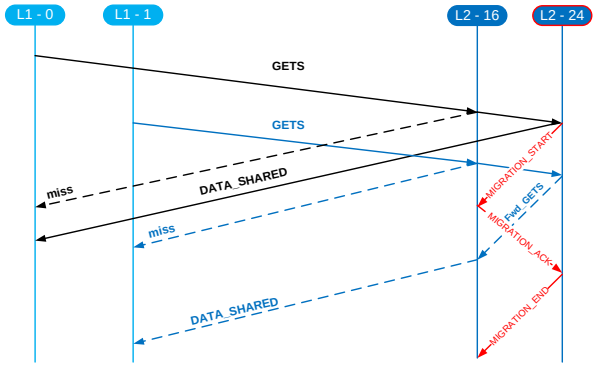


Figure 4: Block migration with the FMA protocol. In case of hit in a L2 bank (L2-24), the copy is sent to the L1 Requestor (L1-0) and a MIGRATION_START message is sent to the L2 Receiver (L2-16). When the L2 Receiver gets the message, allocates a cache line for the copy and replies with a MIGRATION_ACK to the L2 Sender (L2-24), which will conclude the migration process with a MIGRATION_END message. If a new request is received by the L2 Sender while waiting for the MIGRATION_ACK, the request is forwarded to the L2 Receiver, which will serve it.

[2]In the following, we assume that data is the copy of a memory block stored in the cache, that can belong to both the code or data area of a program.

[3]In the following, when we say that a "processor accesses a copy in L2", we assume that the processor tries to access the copy in L1, and, because of a miss, it accesses the copy in L2.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

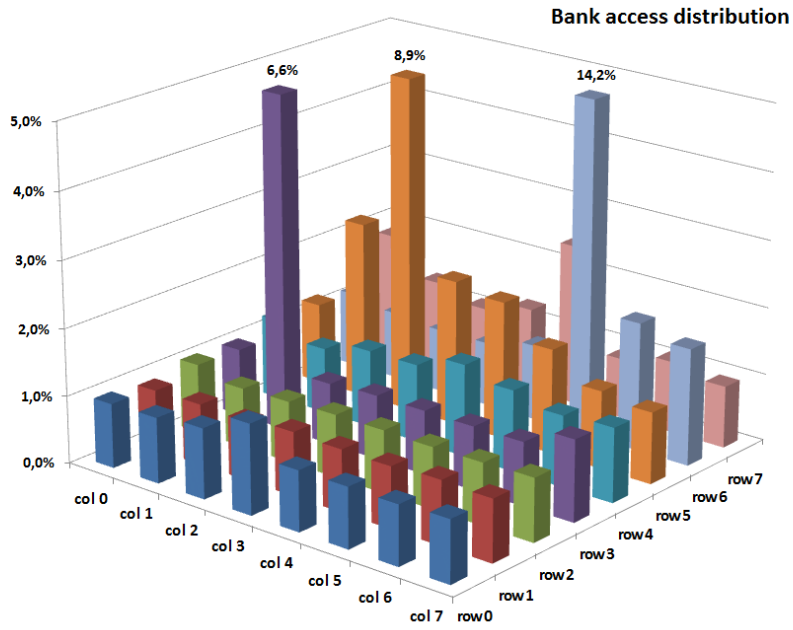


Figure 5: Distribution of accesses to the banks of the SNUCA LLC made by core_7 for the Canneal application.

3 Influence of memory program mapping

A SNUCA cache organization has a fixed mapping from memory addresses to cache banks, so for each address referred by the cores, the target cache bank is the same, independently from the core position in the two considered layouts in Figure 2. Therefore, with reference to Figure 2, half of the cores (core_4 - core_7) changes their location around the LLC when changing from the 8P to the 44P configurations, but such cores continue to access the same banks of the LLC. In this we assume that core X runs the same thread(s) of the application in the two configurations. For example, when core_4 accesses bank_6 in the 8P configuration (Figure 2), the request/response sequence *must cross 6 routers* whereas in the 44P configuration, it *must cross 26 routers*. However, also the opposite is true as there are banks that are accessed in a lower time when changing from the 8P to the 44P configuration (e.g. bank_56, again for core_4). Therefore, the specific memory access distribution, and, in particular, the LLC bank access distribution, of a running application contributes to the performance differences between the two considered layouts. As an example, Figure 5 shows the distribution of accesses made by core_7 to a 44P SNUCA LLC when it executes the *canneal* application from the PARSEC benchmark suite [9]. Accesses are not uniformly distributed among the banks, and in particular, row5, row6 and row7 sum up to more than 50% of the overall accesses count. In the 8P configuration such rows are at the opposite side of core_7, so it is expected that they contribute with a higher average access latency compared to the 44P configuration, with consequent negative impact on performance.

In SNUCAs a memory address is mapped to one specific bank, while in a DNUCA the cache space is partitioned in banksets (column in Figure 2), and each memory address can be mapped to a specific cache line inside each of the banks in a bankset. A copy of a memory location in the DNUCA LLC, migrates towards the side of the requesting core upon a hit. The effect of such migration is the reduction of the access latency to the copy in next accesses, and therefore it induces the automatic reduction of the effects on performance due to a bad program layout. This way, it is expected that the most frequently accessed data move to the lines near to the requesting processor, i.e. line 1 for the 8P configuration and line 1 or 7 in the 44P configuration. However, each layout can have a different impact on the migration mechanism, and so on the expected performance figures. In the 8P layout, all the cores are on the same side, so, once in the cache, a copy can move only in one direction due to a hit. In the 44P layout, if a copy is shared among cores located at opposite sides (for instance due to the sharing of code or fine grain sharing of data), the copy can start moving up and down (conflict hit, Figure 6), possibly never reaching the fastest ways of the cache (Figure 7), with no significant reduction of the access latency. 8P configurations, instead, can further benefit from sharing, as copies reach the fast lines in a shorter time. Conflict hit effect limits performance of DNUCA caches in 44P layout in comparison to SNUCA caches with the same layout [40], [2], and can be a limiting factor even compared

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

to SNUCA and DNUCA caches in the 8P layout. For these reasons, also the distribution of memory accesses and their mapping to the bankset can influence performance in DNUCA, when switching from 8P to 44P layout. The latency of accesses made by core_4 to column 0 decreases when moving from 8P to 44P, and the opposite is true for core_4 and column 7.

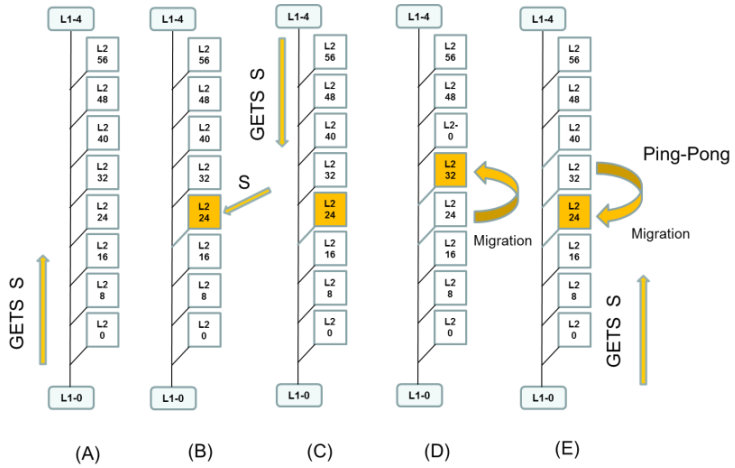


Figure 6: Conflict hits and the ping-pong phenomenon. L1-0 requests data S (A), that is taken from main memory and put in the L2-24 block (B). Then L1-4, located at the opposite side of the cache with respect to L1-0, requests the same data (C). The data now hits in the cache, and migrates towards L1-4 (the requestor) in the bank L2-32 (D). If L1-0 requests again the data S, it migrates towards L1-0, but it does not improve the access latency with respect to (B), nullifying the benefit of migration.

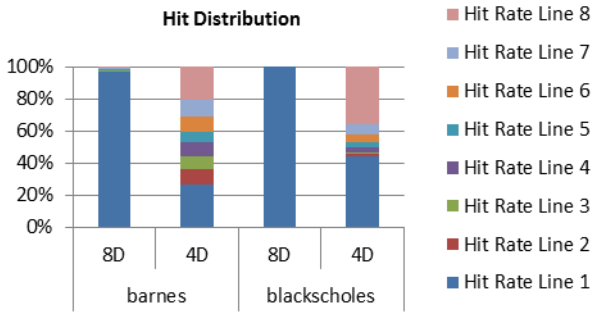


Figure 7: Hit distribution, for the Barnes and Blackscholes applications, in the bank lines. 8D is a DNUCA scheme with an 8P layout, while 4D is the same scheme on a 44P layout. In the 8P layout, both the applications exhibit more than 96% of the accesses to the fastest way (line 1). In the 44P, accesses are spread throughout the lines in different ways for the two applications. It is expected that performance difference when moving from 8P to 44P is higher in Barnes, as accesses in the 44P layouts are almost to all the lines.

3.1 Program memory layout optimizations

The general observation that performance is significantly affected by the access distribution to the LLC banks suggests that specific memory layout optimizations of the application can the average SNUCA access latency. In principle, it is possible to improve the performance of the cache, by mapping the mostly accessed data to the LLC banks near to the requesting cores. For instance, this is what a DNUCA cache tries to achieve in hardware at run-time, promoting the cache lines close to the requesting cores upon a hit event. Other proposals implement this behavior at OS level [45], [46]. A similar approach can also be implemented via software, at compile time, by utilizing a profiling phase which collects information on how accesses are distributed in the LLC, and then using such information to optimize the mapping of data and code to the LLC banks in order to improve overall cache access time or other performance metric. The profiling phase can be performed in two ways: i) (*trace*) collecting the trace of the addresses generated by the application, as generated by the cores towards the LLC, ii) (*bank_count*)

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

collecting the overall accesses count to each LLC bank by each core. The *trace* strategy requires instrumenting and running the application under representative conditions or to simulate its execution in a cycle-accurate, maybe a full-system, simulator implementing all the details of the system memory hierarchy [41], [39], [55]. The *bank_count* approach delivers coarser information than trace-based strategies but can be adopted more easily by adding only specific hardware counters to measure the number of accesses to the LLC banks from each core [44]. Furthermore, the *bank_count* method does not require tool-chain modifications and pre-runs in a simulator.

3.1.1 SNUCA case

For example, as depicted in Figure 8 in case of a SNUCA design, every LLC access by a core is directed to a specific bank according to some specific bits of the address (*orig_bank_id*), specifically from the LLC *index* bits. When remapping is enabled, *orig_bank_id* bits are used to access a remap table and are substituted by the remap table entry value (*remap_bank_id*) before accessing LLC.

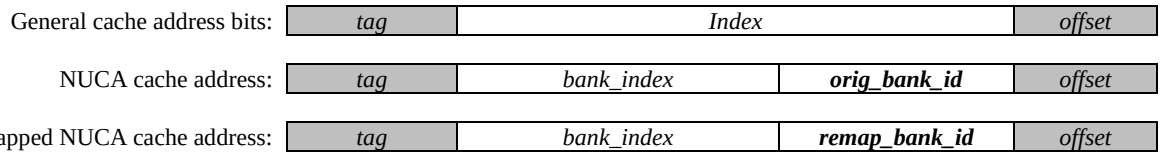


Figure 8: breakdown of the address bits emitted from a core to access (top) a cache in general; (middle) a SNUCA cache; (bottom) a SNUCA cache after bank remapping. In this scheme, the lower bits of the index are used as bank id and for the routing in the NUCA NoC.

In this way, the table can implement a specific bank remapping aimed at reducing the average cache access time seen by the cores. Obviously, remap tables have the same content for all cores to maintain a consistent memory *viewmap*. Operating at a finer granularity (e.g. page level) would need to operate at TLB level and on the virtual memory management modules of the processor [39], [55]. The timing between profile collection and the run of the optimized memory layout is extremely flexible in principle. For simplicity of evaluation in the context of the present work, we assume that an application is run once for profiling and then, when the optimized layout has been calculated, it is re-linked with the remapping information and so successive runs can take advantage of the remapping for better access latency. Other schemes, like optimization during execution or partial execution, are possible but are beyond the scope of the current paper, which sticks to a simple and relatively static profiling approach to focus only on the effects on the 8P and 44P layouts that we are comparing.

We observe that the original *unoptimized* bank access distribution is quite random as it is the result of coding, compiling, linking of application modules and external libraries. Therefore, we have explored also layouts which aim at *worsening* access latency, and thus system performance, so that the performance gap between the worst and best layout can be assumed as an estimate of the maximum *risk* that an application can incur due to a bad mapping.

4 Considered solutions and evaluation methodology

The adopted evaluation methodology can be summarized in the following conceptual steps. Firstly, we have identified the relevant base NUCA organizations, Static-NUCA (SNUCA) and Dynamic-NUCA (DNUCA), to be evaluated in the two considered layouts (*one-side* and *two-sides*, respectively 8P and 44P in our experimental setups), along with specific optimizations for each of them. In particular, the optimized schemes aim to improve the data management policies within the NUCA cache space compared to the standard approach, through improved migration and/or mapping of data within the NUCA cache space. The global objective of these optimized schemes is to improve locality and access latency during NUCA cache access. Overall, we have identified various *NUCA architectures* to be compared. We refer to a *NUCA architecture* (or simply *architecture*) as the instance of a NUCA cache exposing a specific mix of the three facets that we have just introduced: i.e. NUCA organization (SNUCA/DNUCA), layout (one-side/two-sides), and optimizations in the data management policies, as summarized in Table 4.

Secondly, we have compared the performance of such NUCA architectures on a wide variety of complex multi-threaded benchmarks from two widely adopted suites, Parsec [9] and Splash [18], as well as some multi-programmed workloads from SPEC2K.

Lastly, based on the achieved results and on the relative performance of the considered NUCA architectures, we were able to derive some general design guidelines to highlight the most suitable ones as a function of the layout (one-side vs two-sides) as well as of specific performance objectives.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

This kind of work would need to explore a huge space of possible NUCA architectures obtainable as cross-product of all interesting cache parameters (e.g. size and associativity of banks) and all NUCA-specific features even beyond the ones already discussed (e.g. insertion policies upon a miss, number of hits for a DNUCA promotion, and so on). To reduce such space to a limited number of macro-solutions and to try to identify general guidelines, we have aimed at selecting baseline features and parameters in line with state-of-the-art literature on NUCA cache evaluations. Then, for the DNUCA scheme we have considered the well-known, and promising, optimization based on block replication, which is able to tackle the conflict-hit, or ping-pong, issue. While for the SNUCA, we have implemented an application restructuring optimization which aims at limiting possible hit-latency overheads due to random bad memory layouts.

4.1 Basic schemes: SNUCA and DNUCA

In a SNUCA system, each memory block is statically mapped to a unique bank, that also represents the *home node* for that block. The mapping policy between physical address and SNUCA bank is implemented using the low-order bits of the index field (interleaved mapping [5]), in line with Figure 8.

DNUCAs implement a block *migration mechanism* and in our implementation, we consider a *per-hit* [5] block migration: a data promotion is triggered when a block request from an L1 results in a hit in a NUCA bank. In this way, blocks that are frequently accessed by a core are brought in the line of the DNUCA cache that exhibit the lowest latency with respect to that core. As blocks can be stored in *any* of the bankset's banks, we assume a search policy in which L1-to-L2 requests are broadcasted to all the banks of the reference bankset.

4.2 Limiting conflict hits: Re-NUCA

In the 44P configuration, the effectiveness of DNUCA migration mechanism can be limited by conflict hits (or ping-pong), generated by accesses to the same cache block from cores located at opposite sides of the DNUCA cache. The effects of ping-pong are two-folds [40]: performance improvements due to migration are reduced [2], as shared blocks don't succeed in reaching the faster ways, and NoC traffic increases due the useless up and down migration of blocks. As a consequence, also dynamic energy consumption increases [13].

Without additional support, a DNUCA in a 44P configuration may be disadvantaged compared to a corresponding SNUCA cache and to a DNUCA with 8P configurations. To solve such an issue, we consider the Replication-NUCA, Re-NUCA [40], which is a DNUCA cache that allows the replication of shared blocks accessed by cores located at opposite sides of the NUCA, allowing each of them to migrate towards the closest side, as for private blocks. In such a scheme, an invalidation protocol is introduced to guarantee the correctness of the memory operations involving the two copies, so that only one copy of the block will be left on the cache on a write operation.

4.3 Bank level data layout optimizations in SNUCA

In this work, the main objective of the memory layout optimization for the considered multi-threaded applications is to be functional to the fair comparison between 8P and 44P NUCA configurations. Specifically, as the mapping of an application memory image can be, randomly, more favorable to one of the two, we aim at identifying relatively good memory footprints of the applications for both NUCA layouts so that the comparison between the two is not biased by random noise due to the memory-to-cache mapping. From the methodological point of view, we also identify the worst memory-arrangements of applications in the two NUCA layouts, as to assess the maximum performance risk due to randomness in application to cache space mapping. Specifically, the search for bad memory layouts is performed *reversing* the cost function used for identifying the good ones and it essentially looks for higher average hit-latency within the NUCA.

In this paper, we will focus on the NUCA *bank_count* approach for profiling the application behavior. Its main drawback resides in the quite coarse nature of the collected data, as the considered bank size is 256 kB in our case and it could be even bigger in larger LLCs. Another relevant approximation stemming from this choice is that LLC bank access statistics suffers from confusion due to the *modulo effect* in the mapping of the whole memory address space into the cache address space, preventing it from precisely identifying the application parts which are responsible for the collected bank access profiles. These features imply the impossibility of capturing phenomena like *sequences of reuse*[4] and *reuse distance*[5] of addresses which would be crucial to achieve near optimum cache

[4] A *sequence of reuse* can bedefined as the sequence of distinct addresses comprised between two subsequent accesses to a specific memory location *m*. It relates the set of memory locations which are used between successive uses of location *m*, thus in possible conflict with it in cache resources.

[5] *Reuse distance* can be defined as the number of (unique) addresses in a *sequence of reuse* of a memory location *m*.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

behavior [38], [39]. Furthermore, a *trace-based* approach (see Section 4.1), or at least a hybrid approach, could allow the collection of statistics at a finer granularity (e.g. page-level) [39], but for a completely runtime approach, like the one we are aiming at here, employing counters for each virtual memory page would be too costly. In conclusion, the choice of using overall bank access count can be considered a reasonable tradeoff between accuracy and ease of deployment of an optimized application.

In our approach, we assume that the OS can either use offline calculated remap tables, e.g. taking them from the executable binary, or calculating them at runtime after a phase where the running application is profiled. To activate the optimized memory layout, the OS fills a small hardware remap table in each core, similarly to some existing approaches [41], [42], [43]. Specifically, the remap table is accessed from a core before the LLC access (e.g. can be part of the LLC NoC network interface). In the case of 64 banks, the table has 64 entries indexed by the address bits used to identify the target LLC bank (*orig_bank_id_bits*, Figure 8a). Each entry has a $log_2(num\_banks)$ bits content (6 bits in the considered case) which is the remapped LLC-bank position (*remap_bank_id_bits*, Figure 8a) of the bank containing the address issued by the core. Such bits are substituted in the address provided to the NoC network interface so that the standard NoC routing of the remapped address implies that accesses to the banks happen according to the desired remapping. The time and power overheads of the remapping facilities are considered in the presented results.

4.3.1 Optimization algorithm operation

Program layout optimization for SNUCA case is based on cache bank access distribution profile and by information about NUCA cache geometry/timing. From the topology point of view, every cache bank is associated with a 2D-coordinate position inside the NUCA structure (e.g. bank_0 has coordinates (0,0), meaning row 0 and column 0 as shown in Figure 2, which corresponds to a corner of the NUCA NoC). Similarly, also cores are assigned 2D-coordinates and their vertical coordinates correspond to the one of the first row of cache banks (one-side layout) or to the coordinate of either the first or the last bank rows in case of the two-sides layout.

Profile information contains the number of accesses to a cache bank from each core, e.g. $Acc^{(cj)}{}_{bi}$ represents the number of accesses to bank *bi* from core *cj*. Timing features of a specific NUCA architecture encompass bank access time, which is a function of its size, associativity, internal organization and process technology adopted, and the time to traverse a NUCA NoC hop, specifically a link and a router.

The optimization procedure is based on a simplified estimate of the total latency for all the accesses to the LLC. The simplifying assumptions essentially derive from the relatively coarse nature of the profile information on which the optimization relies on and, at the same time, positively contribute to maintain a low computational complexity of the technique. Precise conflict estimation in the NUCA NoC and in the NUCA banks would be possible only if also timing information of each NUCA access would be collected during profile, requiring a *trace_based* profiling approach that allows collecting timing information for each access. Specifically, during the memory footprint optimization procedure the average NUCA access latency of an application run is estimated with the following approximations compared to what a real execution would experiment: a) NUCA NoC traversal is always conflict free in the routers, and b) when a request arrives to the required cache bank, a bank access is immediately possible without wait states due to potential conflicts due to previous accesses from the cores. In principle, these simplifications can lead to optimistic assumptions about the time to access a specific bank from a specific core compared to an actual execution. The estimation error can be expected to grow when the overall average NUCA access frequency increases (thus message density inside the NoC and bank access frequency). However, latency estimations due to each bank, accessed by each core, are all affected by a similar optimistic bias, which allows the comparison of the latency effects of different bank positioning on a relatively fair basis. Essentially, this kind of conflict-free latency estimation enables the building of a topological latency ranking within the NUCA space among the remapped positions that NUCA banks can possibly assume. In practice, the following formula summarizes how the average NUCA access latency is estimated within the bank remapping optimization procedure:

$$Lat(bank\_remap_k) = \frac{1}{Acc\_tot} \cdot \sum_{c_j} \sum_{b_i} Acc^{(cj)}{}_{b_i} \cdot \left[diff\_coord\left(coord(c_j), coord(b_i)\right) \cdot hop\_lat + bank\_lat \right]$$

Where, *diff_coord* function calculates the number of hops needed to go from one position (e.g. from core c_j) to another (e.g. bank b_i) on the NUCA NoC. Specifically, *coord* function returns the coordinates of core c_j and bank b_i in the formula. *hop_lat* and *bank_lat* are the cycle latencies to traverse one hop on the NUCA NoC and to access the target NUCA bank, respectively. $Acc^{(cj)}{}_{bi}$ is the number of accesses performed by core c_j to bank b_i, and constitutes the overall weight of the approximate latency seen by such core to access such bank one time, considering the positioning of the specific core (which is also a function of the one-side/two-sides NUCA layout) and of the specific

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

bank. Then, *Acc_tot* is the total number of accesses, which allows to obtain the estimated average access latency. The overall average latency is a function of the specific bank remapping[6] $bank\_remap_k$, which is an assignment between each original bank and its remapped position in the NUCA cache space. In fact, the remapped coordinates of each bank affect the hop count of every access to the bank from every core.

The objective of the optimization is to minimize *Lat* identifying the proper bank remapping, i.e. permutation of bank coordinates. From the practical point of view, in building the ranking between different remappings (e.g. $bank\_remap_k$ and $bank\_remap_j$), the addition of *bank_lat* and the division by *Acc_tot* can be omitted as they do not alter the relative order of remappings.

Here we make a few observations to highlight how the identification of remapped bank positioning is far from trivial. Firstly, moving one bank one-hop closer towards a core, causes a decrease in the estimated latency between such core and bank. However, also the paths between other cores and such banks are changed too, and some of them are shortened by one and some others are one hop longer. Therefore, estimated latency components from some cores are decreased and others are increased. Consequently, it is not obvious how the overall latency estimate of the application will be affected by this bank mapping change. In principle, after the remapping of such banks a net advantage can be expected if the cores that are closer to the bank perform a higher number of accesses than the ones that see a longer distance. Typically, all cores access all banks with a specific access count and thus, focusing on individual bank position related to only one core can be misleading. The identification of good strategies for seeking good bank remappings need to wisely consider the aggregate access profile to the banks by each core. Unfortunately, the complete enumeration of all possible remappings of NUCA banks in order to identify the minimum achievable latency is completely not viable due to the extremely high number of possibilities (e.g. $1.27x10^{89}$ for just 64 banks).

Therefore, despite the highlighted intrinsic difficulty in deciding the remapped position of each bank in insulation from the others, computationally affordable strategies need to actually be conceived as to process one bank at a time. In this sense, our heuristic optimization, and search, strategy works in a *one-pass* (not iterative) and *constructive* fashion, which aims at deciding the new position of one bank after the other, each in a promising place among the free ones left. In this way, the last considered bank will then have only one place left to go. In this process, each bank is conceptually positioned according to its access profile from all the cores and, consequently, to its corresponding contribution to the overall estimated average access latency of the application according to the above described formula. Specifically:

a) for each bank b_i we calculate its overall average latency contribution for every possible coordinate of the NUCA matrix (in our case, 64 possible positions). Latency contribution is evaluated according to the above described formula for the sole specific bank;
b) we define min_lat_i and max_lat_i as the minimum and maximum latencies that bank b_i can induce and we sort possible bank positions in ascending latency order;
c) then, banks are considered one at a time for placement according the (max_lat_i - min_lat_i) value, in reverse order. In practice, the first bank to be placed will be the one with the highest difference between the maximum and minimum induced latencies as a function of its possible positioning. Possible positions of the considered bank are evaluated in latency order (see step a)), and the first free (unassigned) coordinate is assigned.

The idea behind the bank order in placement is that a bank that shows similar latencies when varying its position in the LLC is not very critical in its placement. In an extreme situation, wherever is placed, it will deliver the same overall access latency when considering the accesses from all cores. So, it can be safely placed later, when some more critical positions are already taken. This is a crucial observation because this algorithm determines the optimized memory-to-cache mapping essentially inducing a permutation of the NUCA content at bank granularity. Therefore, every bank content in the original memory layout must have a distinct NUCA cache position in the optimized memory layout. From the opposite perspective, a bank showing big latency differences for the various NUCA positions, can benefit from early positioning to exploit the freedom to assign it to the most rewarding coordinates. In fact, after some banks are assigned, their corresponding coordinates cannot be used anymore, reducing progressively the placement flexibility of subsequent banks, up to the last one which will be assigned to the only one residual position.

[6]Every LLC bank *remapping* corresponds to a specific permutation of the original bank position in the NoC topology.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

We identified and tuned our heuristic remapping algorithm with the aim to be very simple and quick to run, in order not to pose specific constraints to its adoption also in contexts where it needs to be used even at runtime. We expect that a more complex algorithm (e.g. iterative or Branch-and-Bound-like), could reach even improved results [38], investing significantly more computational time in an offline optimization process. However, based on our past experience [38], [39], [54], in this work we evaluated various algorithms, and algorithm variations, for our simple approach and the described one provided the best and most stable results across the considered benchmarks.

4.3.2 Worst case memory-to-cache mapping

We underlined in the previous sections that the mapping between the memory footprint of an application and the cache space of a processor (e.g. Last-level NUCA in this case) is affected by significant mapping randomness, and this determines a degree of randomness in the average cache latency performance, which motivates the optimization procedure. Anyway, without optimization, the memory to cache mapping is out of the control of the programmer, therefore we have identified the worst memory-to-cache layout at the bank-level granularity in order to quantify the maximum performance risk that could emerge running an application onto a NUCA cache, as it could possibly result from a specific compilation-execution. To do so, we used the same algorithm that looks for the good mappings but with the *reversed* objective in the cost function. Specifically, banks are considered in the same order, from the most to the least critical in the positioning, but they are assigned to the free position that induces the maximum, instead of the minimum, overall average latency from all cores. This way, it is highly likely that the most latency-sensitive banks can be positioned in their worst position. Doing so, subsequent banks will have less placement freedom but are less sensible to their positioning.

Results show that the maximum risk coming from a random memory-to-cache layout is quite high and thus, the original application could behave even quite worse than what was observed in sample experiments. As a side note, these results make a strong case for the wider adoption of memory-to-cache optimization algorithms in order to limit the high cost of potential bad mappings.

4.3.3 DNUCA optimization strategies

A main memory location is mapped into the DNUCA space according to two principles. Firstly, its absolute address implies which *bankset* (see section 3) such location will belong, e.g. the "bank column" in Figure 2 and Figure 6. This mechanism is similar to what happens in a SNUCA cache, in which however both the column and the row are determined by the memory address. Diversely from SNUCA, a DNUCA cache allows the migration of cache blocks among banks in the same bankset according to a hardware policy. Therefore, the maximum ability for a memory-to-cache optimization strategy operating on the memory footprint of an application is to decide the bankset of each memory positioning unit (e.g. banks or bankset). In our cases, the number of banksets is typically only 8, and up to 16, compared to the corresponding 64 and 256 banks of the SNUCA counterparts. Therefore, the optimization flexibility is quite smaller than what is potentially available in the SNUCA case for two reasons. Firstly, the maximum number of layouts achievable through the permutation of 8 banksets is only 8! (compared to 64! for a similar SNUCA). Even if we would be able to exhaustively pick the one delivering the best estimate of the average access latency, the bigger placement granularity (bankset instead of bank) forces the algorithm to perform a quite coarser, thus less effective, tuning. Furthermore, the placement algorithm has no control over what happens inside the bankset, which is fully delegated to the hardware DNUCA policy operation.

For these reasons, we can expect a memory-to-cache optimization in DNUCAs far less effective than in SNUCAs. We implemented such an exhaustive algorithm, which operated exactly as the SNUCA one but at bankset level, and results indicated really marginal improvements for the considered benchmark suites. Also the worst-case mappings were quite close to the average and best-case versions. Therefore, we decided not to consider any software optimization in the DNUCA case as it showed not to be heavily affected by potential non-determinism in the memory organization of the applications.

5 Experimental setup

We performed full-system simulations using Simics [14]. We simulated an 8-cores 64-bit CMP system, where each core is in-order and running at 4 GHz. We performed a portability analysis considering also 16-core systems. We utilized GEMS [15] to simulate the cache hierarchy where the shared NUCA L2 cache is composed by 64 banks (each of 256 KB, 4-way set associative, sequential access), for a total storage capacity of 16 MB. In our analysis, the SNUCA and DNUCA architectures implement a directory version of MESI, in which the directory is *non-blocking*[10] and distributed. By adopting an inclusive cache hierarchy, we avoid the need of a centralized directory, so that directory information is held in the copies stored in L2 banks. Cache latencies to access TAG and TAG+Data

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

have been obtained by CACTI 6.5 [16], [33]. The NoC is organized as a partial 2D mesh network, with 64 wormhole [7] switches (one for each NUCA bank); NoC switch and link latencies and consumption have been calculated using the Orion estimators [34], [35] and the Berkeley Predictive Model [17]. Table 2 summarizes the configurations for the considered CMPs and Table 3 shows the main power consumption parameters of the NUCA cache sub-modules.

Our simulated platform runs the Sun Solaris 10 operating system where we run applications from SPLASH-2 [18], PARSEC 2.0 [9] multithreaded benchmark suites, as well as multiprogrammed workload from SPEC2K [73]. A brief description of the nature of each considered benchmark is summarized in Table 1. Detailed characterization of the SPLASH and the PARSEC suites, and of the associated communication and sharing patterns, can be found in [71], [36]. Essentially, benchmarks from such suites implement diverse parallel processing approaches in their multi-threading execution (i.e. work splitting among threads) and expose a wide range of synchronization and sharing patterns, from fine grain and dense, to coarse grain and less dense. For these reasons, they are representative of heterogeneous workloads for modern and future multi-core systems. All the applications were compiled with the *gcc* version provided in the Sun Studio 10 suite. The simulations comprise the execution of the parallel region of interest of the benchmarks, with a warm-up phase of 50 million instructions. We simulated the execution of *ocean*, *barnes*, *radix*, *radiosity* and *raytrace* from the SPLASH-2 suite (in the following SPLASH), and *blackscholes*, *bodytrack*, *canneal*, *streamcluster* and *swaptions* from the PARSEC 2.0 suite. Furthermore, we also run a set of applications from the SPEC2K suite (mcf, bzip2, art, and parser, each loaded twice in order to have eight processes) as a sample of multiprogrammed workload. We call Parsec & Spec the subset constituted by PARSEC 2.0 and SPEC2K applications. We choose the simulated *Execution Time* as the reference performance indicator, and we utilize the average L1 miss latency (Figure 12) and the distribution of hits within the NUCA lines defined in Figure 2 (Figures 13, 14 and 15) to investigate and explain the performance differences among the configurations. Then, overall energy figure is adopted as a consumption metric of the considered configurations and, along with the breakdown into static and dynamic components, it allows the investigation into the behavior of the considered configurations.

As for the experiments, we considered, for the two layouts of Figure 2, the different SNUCA and DNUCA architectures with layout optimizations and data management policies according to table 4, where we adopt the notation X_Y to denote that application X is running on the system architecture Y.

Table 1: Considered applications from Splash-2 and Parsec workloads.

SPLASH-2	
Barnes	N-body Simulation
Ocean	Ocean Current Simulation
radiosity	Graphics
Radix	Integer Sort
raytrace	3D Rendering
PARSEC 2.0	
blackscholes	Financial Analysis
bodytrack	Computer Vision
canneal	Engineering
streamcluster	Data Mining
swaptions	Financial Analysis
SPEC2K	
Mcf	Combinatorial Optimization
bzip2	Compression Algorithm
Art	Image Recognition / Neural Networks
Parser	Word Processing

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

Table 2: Configuration parameters of the considered CMP architectures.

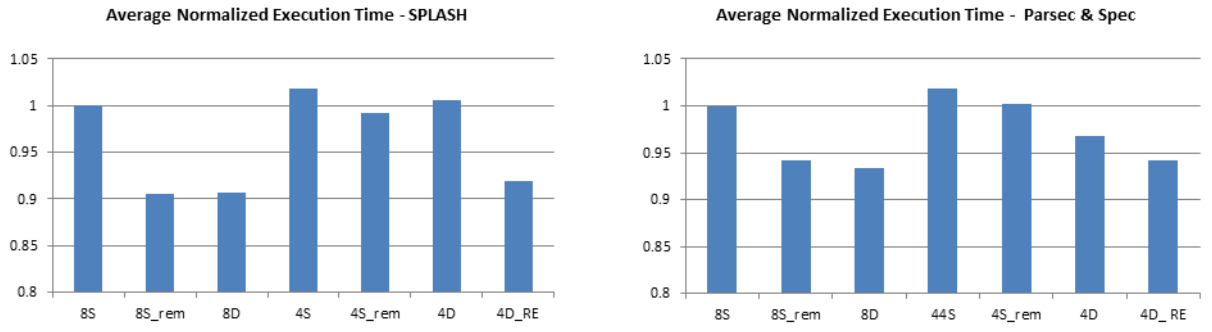
Processors	*8 cores (64-bit), connected all to one side (8P) or four to each of two opposite sides (44P)*
Clock Frequency/tech.	*4 GHz / 32 nm*
Coherence Scheme	*Directory based MESI*
L1 cache	*Private 16 kBytes I + 16 kBytes D, 2 way set associative, 1 cycle to TAG, 2 cycles to TAG+Data*
L2 cache	*16 MBytes, 64 banks, 64 bytes block*
L2 cache bank	*256 kBytes, 4 way set associative, sequential, 2 cycles TAG, 5 Cycles TAG+Data.*
NoC configuration	*Partial 2D Mesh Network; 256 bit flit; NoC switch latency: 1 cycle; NoC link latency: 1 cycle*
Main Memory	*2 GBytes, 240 cycles latency*

Table 3: Energy parameters of the NUCA cache sub-modules.

Cache bank static power [mW]	*95.01*
Cache bank dynamic energy (Hit) [pJ/access]	*333.43*
Cache bank dynamic energy (Miss) [pJ/access]	*5.95*
Switch static power [mW]	*66*
Switch dynamic energy [pJ/flit]	*127*
Link energy [pJ/flit]	*58*
Energy per main memory access (nJ/access)	*12.25*

Table 4: Summary of the experiments performed and related configurations.

Name	*Layout*	*NUCA Organizations*	*Optimizations*
8S	8 cores at one side of the cache (8P)	SNUCA	None
8S_rem	8 cores at one side of the cache (8P)	SNUCA	Software mapping of the mostly accessed memory blocks near to the cores using them (Section 4.3)
8D	8 cores at one side of the cache (8P)	DNUCA	None
4S	8 cores, 4 at one side of the cache, 4 at the opposite side (44P)	SNUCA	None
4S_rem	8 cores, 4 at one side of the cache, 4 at the opposite side (44P)	SNUCA	Software mapping of the mostly accessed memory blocks near to the cores using them (Section 4.3)
4D	8 cores, 4 at one side of the cache, 4 at the opposite side (44P)	DNUCA	None
4D_RE	8 cores, 4 at one side of the cache, 4 at the opposite side (44P)	DNUCA	Replication of a copy once accessed by cores at the opposite sides of the LLC (section 4.2)



(a) **(b)**

Figure 9: Average Normalized Execution Time for the set of SPLASH (a) and Parsec & Spec (b) applications.

doi: 10.1016/j.future.2017.06.001. © Elsevier

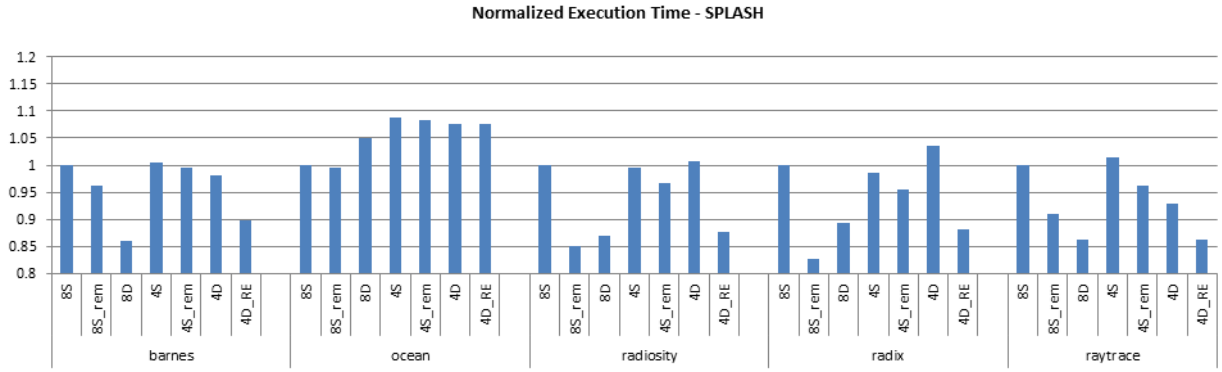


Figure 10: Normalized Execution Time for the SPLASH applications.

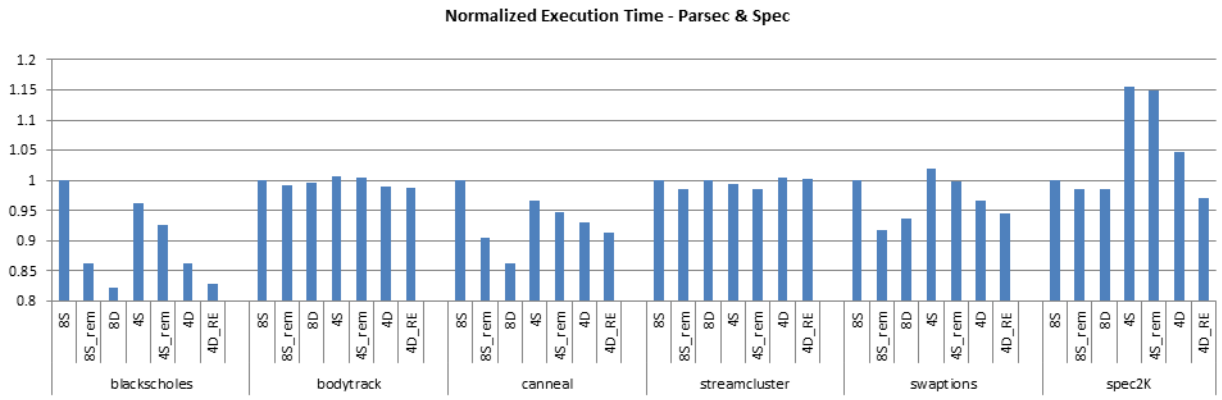


Figure 11: Normalized Execution Time for the Parsec & Spec applications.

6 Results

In this section we analyze the performance, network traffic and energy metrics of the considered NUCA architectures. Furthermore, we discuss the resulting design guidelines for one-side and two-sides layouts and we highlight the portability of results on different architectures.

6.1 Performance evaluation

Figures 9-a and 9-b show the average Execution Time for the two subsets of benchmarks (SPLASH and Parsec & Spec) running on the system architectures described in Table 4, by assuming the 8S configuration as a baseline.

The DNUCA cache on the 8P layout (8D) is the most performing solution for all the benchmarks. The Execution Time reduction is about 10% for SPLASH applications (Figure 9-a) and 7% for Parsec & Spec ones (Figure 9-b). When considering individual applications, DNUCA 8D can achieve substantial performance improvements: up to 15% for some SPLASH-2 applications and 20% for some Parsec & Spec applications (Figure 10 and Figure 11). Conversely, in Ocean there is a performance degradation due to L2 miss rate increase originating from a higher number of conflicts in the entry point banks, which are the banks where memory blocks are placed in case of a miss. Specifically, the blocks in the entry point bank are often evicted before they are re-accessed by the threads, thus avoiding promotions and triggering additional misses. The latency gain due to the migration mechanism is then nullified by the increase in the L2 miss rate (Figure 12). So, the migration mechanism of DNUCA is not fully adequate for Ocean's locality, as also observed in [6]. The miss rate increase in Ocean can be observed also in SNUCA caches, when moving from an 8P to a 44P layout. This can be explained by considering that Ocean has a high reuse of shared data [36], and changes in the temporal behavior of the system (determined by the different position of the cores) can produce different sequences of conflicting accesses to such data and consequent changes in the number of misses.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

In the 44P layout, DNUCA caches get lower gains than in the 8P one (on average, 1% for SPLASH and 4% for Parsec & Spec). This is a consequence of the conflict hit phenomenon, which in many applications does not allow the migration of code and data to the faster ways, resulting in a significant distribution of accesses on all the cache lines, and not only on the faster ones (Figure 13). For example, consider the Barnes and Blackscholes applications. In Barnes_8D, 99% of the accesses are to the fastest way (Figure 13), unlike Barnes_8S (Figure 14), where 40% of the accesses are in a central line (line 4), and Barnes_4D (Figure 13), where, although the number of accesses in the faster ways increases, the fraction of accesses to slow lines is still not negligible. Accordingly, the performance increase of Barnes_8D (15%) is not observed in Barnes_4D (3%). In Blackscholes, for both 8D and 4D, the hit rate in the faster ways increases compared to the 8S configuration, resulting in a performance gain of 18% and 14%, respectively. Conflict hits may have also a negative impact on performance in the 8S configuration. For Raytrace application in the 8S configuration, most of the hits are concentrated in a central line (line 5). In the 4D, the spreading of accesses over the lines increases the hit latency (Figure 12) with consequent performance degradation.

The adoption of a DNUCA architecture does not produce significant performance variations for Bodytrack and Streamcluster. This is due to the low exhibited L1 miss rate that makes the applications substantially insensitive to L2 latency. For SPEC2K, in the 8P layout, the distribution of accesses has a high number of hits in the faster lines already in the SNUCA, so there are no significant improvements by introducing optimization or a DNUCA (Figure 15). In the 44P layout, instead, the distribution produces a performance degradation in SNUCA based solutions, a partial improvement in the DNUCA architecture, and big improvements with a Re-NUCA architecture mainly thanks to the sharing of code in the benchmarks.

In the 8P configuration, performance similar to a cache DNUCA can be achieved with a simpler SNUCA architecture by adopting the data layout optimization presented in section 4.2. On average, the Execution Time on SNUCA with layout optimization is like the DNUCA for the SPLASH application, and 2% bigger for the Parsec & Spec applications (Figures 8 and 9). Basically, the software optimization replaces the hardware migration mechanism of DNUCA, and maps the mostly used memory blocks on the faster cache banks. Anyway, information on the banks usage are static, while in DNUCA bank usage (or, better, the frequency of accesses to the banks) is exploited at runtime, and this makes the DNUCA more adaptive to the dynamic behavior of the workload. Besides, the software remapping operates also among the horizontal banks of the cache (with reference to Figure 2), i.e. it tries to improve the latency moving banks not only along columns, but also along rows (a similar 2D migration mechanism has not been implemented in DNUCA, as it makes too difficult the management of coherence and search [6]). The overall effect is that DNUCA performs slightly better in some cases and the software remapping solution has the edge in some others. In particular, this situation can be observed for the Radiosity, Radix and Streamcluster applications. For instance, by focusing on Radiosity application, data of Figure 13a indicates that almost 99% of the accesses are in line 1 for the 8D, while, from Figure 14a, only 77% of the accesses are at the same line in 8S_rem. However, 8S_rem presents the lower L2 access latency and the lowest Execution Time. This is because, due the horizontal optimization performed in the 8P_rem configuration, the resulting access latency to the fastest NUCA lines is always smaller than to a DNUCA (Figure 16), and the overall contribution of such lines to the latency is smaller than in DNUCA (Figure 17). The same considerations hold for the other applications.

Overall, SNUCA data layout optimizations are more effective in the 8P configuration than in the 44P configuration. In general, in the 8P configuration with remapping, moving banks closer to the requestors (vertical movement) always improves the access latency, while the horizontal moving improves performance if it is in the direction of the most requesting cores. In the 44P configuration, the gain of moving a bank is always reduced by the accesses made by cores located at the opposite side. Figure 18 and Figure 19 show such effects, in the case of the Barnes application. In the 8P configuration the access latency significantly increases when moving to the slowest lines, with a 2.5x ratio between the access latency of the fastest and the slowest lines, while in the 44P configuration the access latency of the central lines is almost constant. Only for the DNUCA the fastest lines (line 1 and line 8) show a decrease of the access latency of 54% and 33% respectively. Software remapping brings also a reduction of network traffic and dynamic energy consumption, as detailed in the following sections.

In a 44P layout, significant performance improvements in DNUCA can be achieved by adopting, together with migration, a replication policy that limits the effect of ping-pong (conflict hits). With such solution (4D_RE), on average, 44P layout achieves performance similar to the 8D one (Figure 9-a and 9-b). In particular, for each application, 4D_RE outperforms all the other solutions based on a 44P layout (Figure 9-b and 10), with the exception of Streamcluster, where the most performing solution is the 4S_rem, although the difference is limited due to the low sensitivity of the benchmark to L2 access latency. With the reduction of conflict hits, (replicated) copies can move towards the requesting cores so that most of the accesses are in the faster lines (Figure 13a-b) with

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

consequent reduction of the L2 access latency (Figure 12a–b). Performance improvement is achieved at the cost of added complexity in the coherence protocol, that must support replication and invalidation of copies [40], and that can cause an increase of both network traffic and dynamic power consumption. Such effects will be evaluated in the next section.

In the case of SNUCA cache, there are no significant differences in performance, on average, when the applications are executed in a 44P layout or in a 8P layout (2 % for the SPLASH subset, Figure 9-a, and less for Parsec & Spec, Figure 9-b). Anyway, we can highlight that, if the machine is devoted to the execution of a single application, the layout may have an impact on performance, as can be seen by comparing the behavior of some applications when running in a 8P or in a 44P system layout: for Ocean, Execution Time increases by 9% (Figure 10), for Blackscholes the Execution Time decreases by 4%, for Canneal decreases by 4% and for SPEC2K there is an Execution Time increase by 15% (Figure 11). For Ocean, as previously discussed, this is consequence of an increased miss rate. For the other applications, the miss rate does not change significantly, and it is instead the specific distribution of accesses to the lines that generate the difference.

These results highlight a different sensitivity of performance to the application behavior (in terms of sequences of addresses generated and, then, of distribution of accesses) in the 8P and 44P configurations. To investigate such sensitivity better, we have explored also layouts which aim at *worsening* access latency, and thus system performance, in order to estimate the maximum performance *risk* for an application due to a bad memory layout. Figure 20 shows the ratio between the worst and the best achieved Execution Time, for the SPLASH (a) and Parsec & Spec (b) benchmarks. Results indicate that the 44P floor-plan is less sensitive to data layouts, although the best performance improvements can be achieved with the 8P layout, and agree with Figure 18 and 19: the lower variation in the access latency of the 44P layout determines fewer opportunities for improvements or deteriorations in performance and, then, less sensitivity.

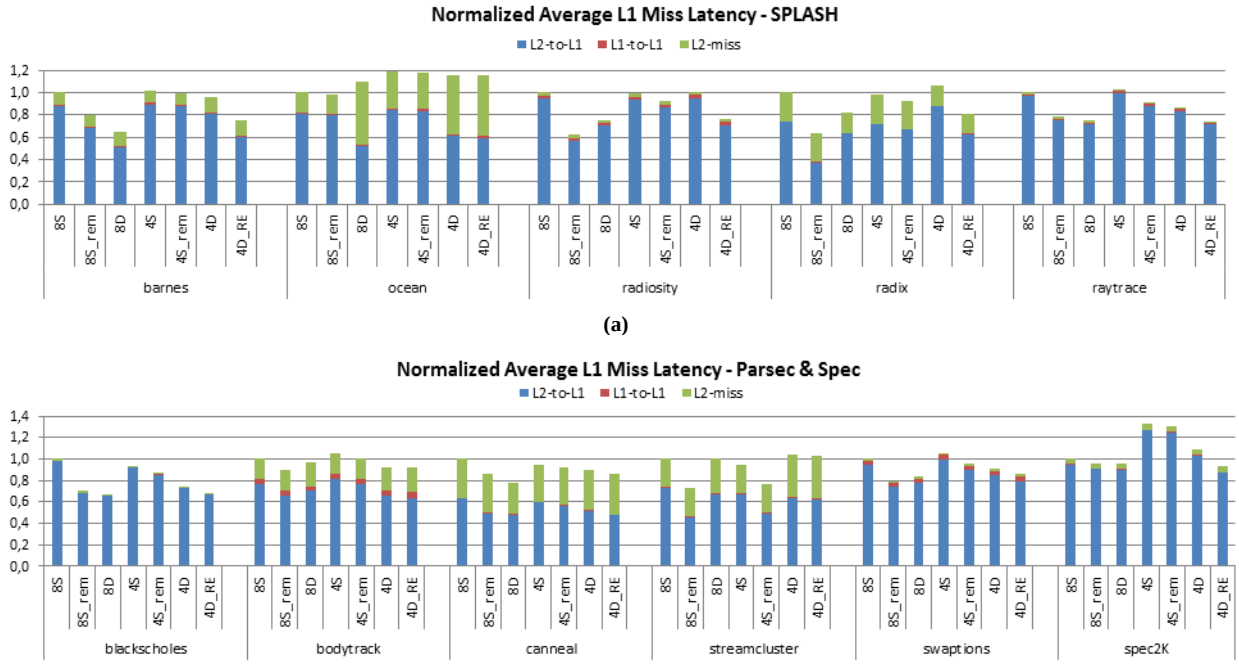
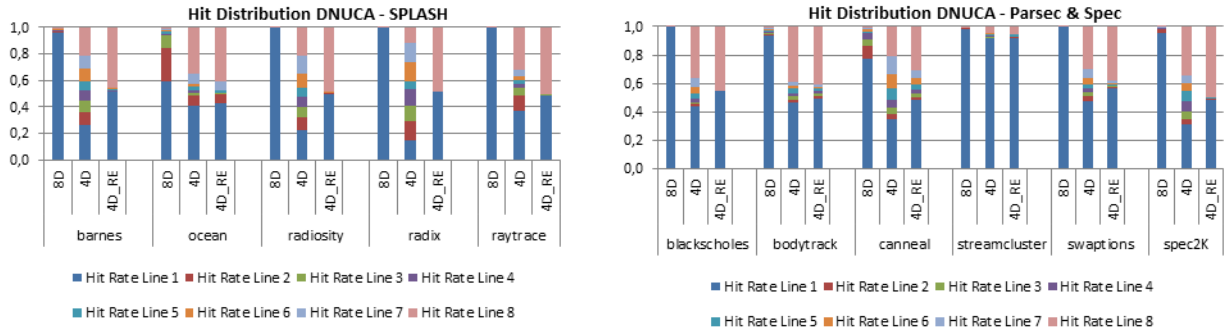


Figure 12: Breakdown of the normalized average L1 miss latency, for SPLASH (a) and Parsec & Spec applications (b), with respect to the components L2 miss (green), L2 hit that determines a transfer from L2 to L1 (blue), and L2 hit that determines a transfer from L1 to L1 (red).

doi: 10.1016/j.future.2017.06.001. © Elsevier



(a) (b)

Figure 13: Hit distribution in DNUCA lines, for the SPLASH (a) and Parsec & Spec applications (b).

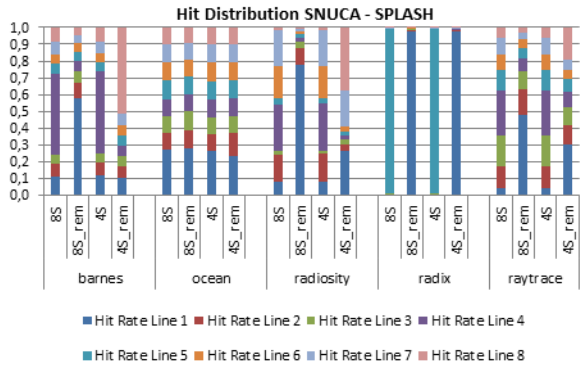


Figure 14: Hit distribution in SNUCA lines, for the SPLASH Applications.

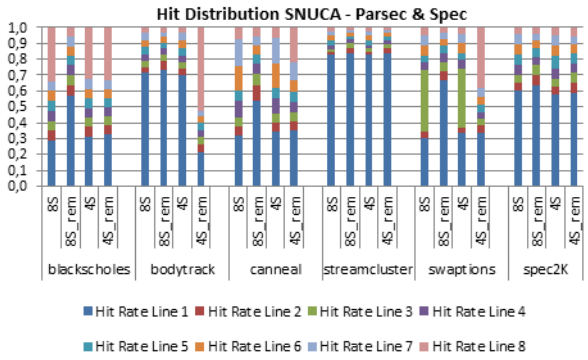


Figure 15: Hit distribution in SNUCA lines for the Parsec & Spec Applications.

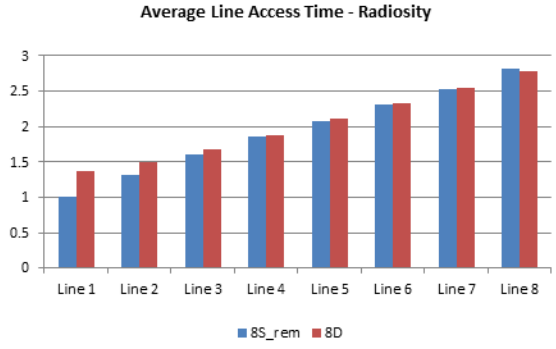


Figure 16: Average Line Access Latency of the NUCA cache line for the Radiosity application, when running in the 8D or 8S_rem configuration.

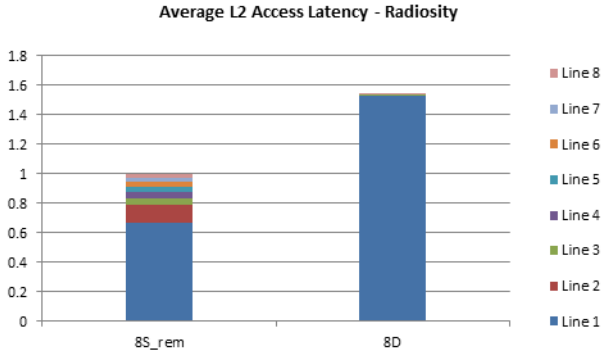


Figure 17: Breakdown of the L2 Access Latency for the Radiosity Application, when running in the 8D or 8S_rem configuration. Data represents the contributions of the single lines to the access latency.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

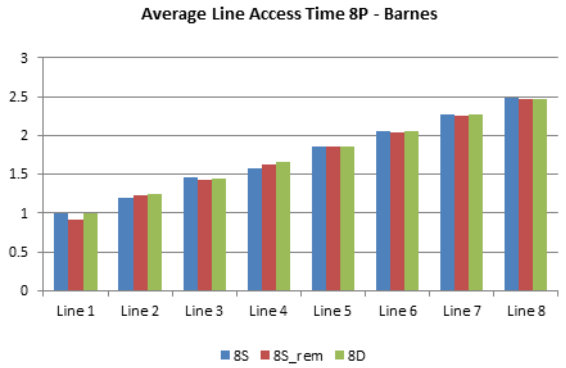


Figure 18: Average Access Latency of the NUCA cache line for the Barnes application, when running in the 8S, 8D or 8S_rem configuration. The access latency significantly increases when moving to the slower lines, with the ratio fastest/slowest that is around 2.5.

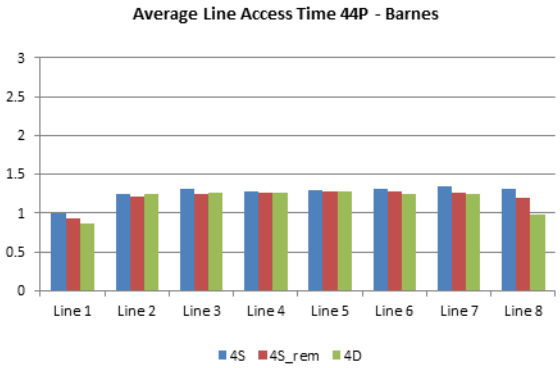
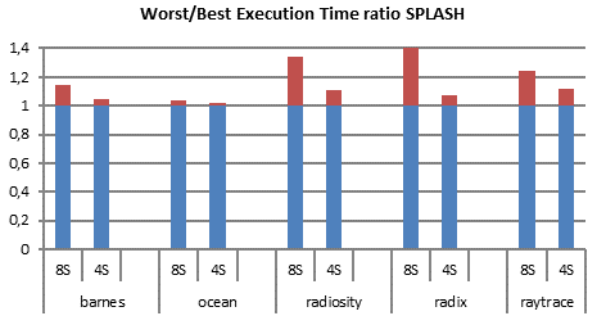
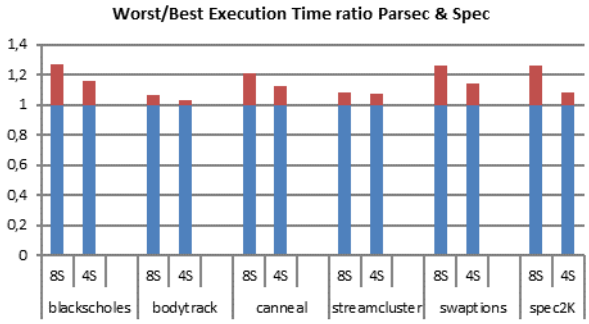


Figure 19: Average Access Latency of the NUCA cache line for the Barnes application, when running in the 4S, 4D or 4S_rem configuration. The access latency of lines from 2 to 7 is almost constant, while it is reduced mainly for the 4D architecture.



(a)



(b)

Figure 20: Execution Time increase (in percentage) between the best and the worst performing mapping layout for the SPLASH (a) and Parsec & Spec (b) applications. The 44P layout is less sensitive to data layouts, although the best performance improvements can be achieved with the 8P layout.

6.2 Network traffic and energy consumption

In this subsection, we analyze the network traffic and energy figures of the considered configurations in order to complement the performance analysis discussed in the previous sections.

6.2.1 Network traffic

Figure 21 and 22 show the network traffic for the SPLASH and Parsec & Spec benchmarks, highlighting the components due to control (requests and coherence management packets) and data (answers that carry data) traffic. The network traffic of the 8S configuration is taken as reference for each application.

In both benchmark families, DNUCA solutions have the highest network traffic. In particular, a reduction of the traffic related to data messages can be observed in DNUCA 8D and 4D, as the migration mechanism brings data closer to the requesting core, but there is also a substantial increase in the control traffic, due to the search mechanism. The overall effect is a significant increase in the network traffic (up 2 times the traffic of the reference system). Besides, the reduction of the data traffic is less evident in the 4D configuration, due to the limiting effects of ping-pong.

Solutions based on remapping induce the lowest traffic, with a reduction of both control and data traffic. Thanks to remapping (8S_rem), on average data are placed closer to the requesting cores, and this reduces both control and data traffic, as it reduces the number of hops that must be traversed to reach the banks. Once again, the reduction is less evident in the 4S_rem configuration, due to the limits highlighted in the previous section.

In the replication-based solution (4D_RE), there is a decrease in network traffic over the corresponding 4D one. 4D_RE are based on a more sophisticated coherence protocol than DNUCA. Such protocol, indeed, must keep

doi: 10.1016/j.future.2017.06.001. © Elsevier

coherent not only copies in the L1 caches, but also the copies in the L2 cache. This can determine an increase in the network traffic. However, 4D_RE succeeds in limiting the ping-pong effects, and, via replication, let mostly accessed block to reach faster ways, also if they are shared (and this holds for both data and code). As a consequence, there is a reduction in the number of routers and links that must be traversed by requests, and then a reduction in network traffic.

In SNUCA based systems, the distribution of accesses determines also the network traffic. In fact, the shape of the distribution induces the average distance of accessed data and thus induces both the average NUCA access latency and the amount of traversed NoC hops, i.e. network traffic. So when switching from 8S to 4S, a reduction of L2 access latency corresponds to a reduction of network traffic (for instance in Blackscholes and Canneal, Figure 22 and Fig 12-b).

In all DNUCA based solutions, most of the hits are in the fastest lines, i.e. line 1 in the 8P layout, and line 1 and 8 in the 44P layout (Figure 13 a and b). Such effect can be exploited to reduce network traffic by using a smart search policy. We adopt a smart search policy that a) first searches the block in the fastest way, and b) in case of miss, broadcasts the search to the other banks. Such policy should reduce network traffic, as most of the hits are in the fastest ways, and can increase performance due to the consequent reduction of network congestion, but increases the hit latency for lines different from the first, and increases the miss detection time, with a consequent negative impact on performance. Results (Figure 24) indicate that DNUCA 8P (8D) and Re-NUCA in 44P (4D_RE) with smart search present an increase in execution time, with respect to a solution without smart search, that is always below 1%. In DNUCA 44P (4D), the ping-pong effects prevent most used blocks reaching the fastest ways, and this limits the effectiveness of the smart search (the Execution Time increases up to 3.5%). Figure 23 shows that, except for 4D, the network traffic becomes comparable to the SNUCA one.

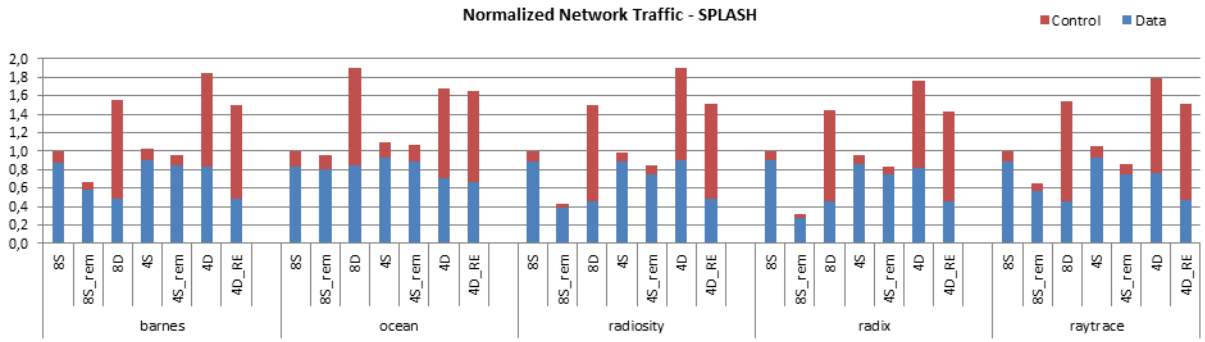


Figure 21: Normalized NoC traffic for the SPLASH applications. Control messages are composed by 8 bytes (header only). Data messages are composed by 72 bytes (8 bytes for the header, 64 bytes for the data).

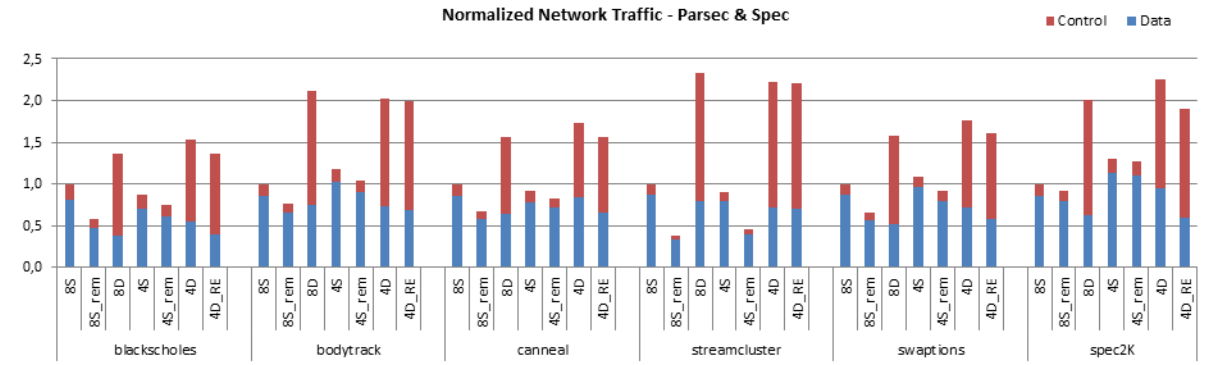
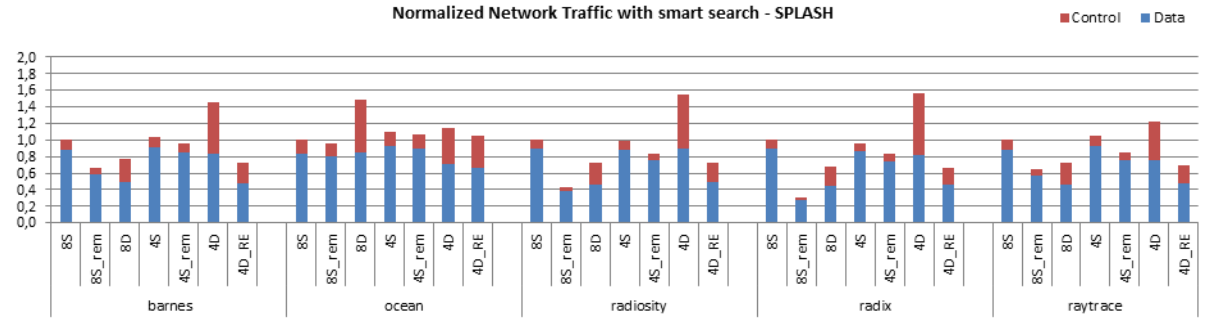
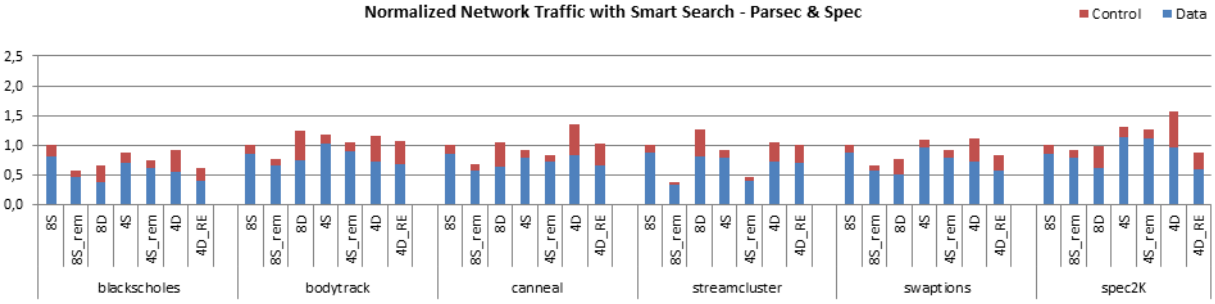


Figure 22: Normalized NoC traffic for the Parsec & Spec applications.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier



(a)



(b)

Figure 23: Normalized NoC traffic for the SPLASH (a) and Parsec & Spec applications (b), when adopting a smart search mechanism for the DNUCA.

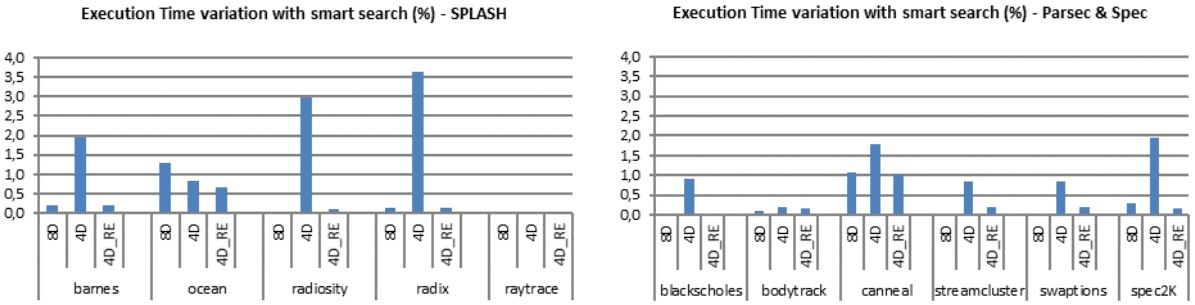


Figure 24: Execution Time variation (in percentage) when adopting a smart search mechanism for the DNUCA. The differences are below 1%, and higher for the configurations in which the migration mechanism is not performing well (44P configurations).

6.2.2 Energy consumption

Energy consumption can be classified in dynamic and static. Dynamic energy consumption is determined by the switching activity of the circuits, while the static one is determined mainly by leakage currents. As for the dynamic energy consumption, it is composed by the energy to access the cache, the energy spent in transmitting data over switches and routers, and the energy to access the main memory in case of LLC misses. For this evaluation, we adopt the methodology presented in [29] and for all DNUCA solutions we consider the versions optimized with *smart search*.

Figure 25 presents the dynamic energy consumption for SPLASH and Parsec & Spec benchmark sets. The components due to cache accesses, links and routers are highlighted. As in previous sections, we assume the 8S energy for each application as the baseline reference for result normalization.

8S_rem solution delivers the lowest dynamic energy consumption for all the benchmarks. In such a solution, the effectiveness of remapping allows data to be near to the requesting cores, thus reducing the number of routers and links that must be traversed by the blocks and then the overall dynamic energy. The component due to the cache banks does not change, as the remapping algorithm does not alter the number of cache accesses. The

doi: 10.1016/j.future.2017.06.001. © Elsevier

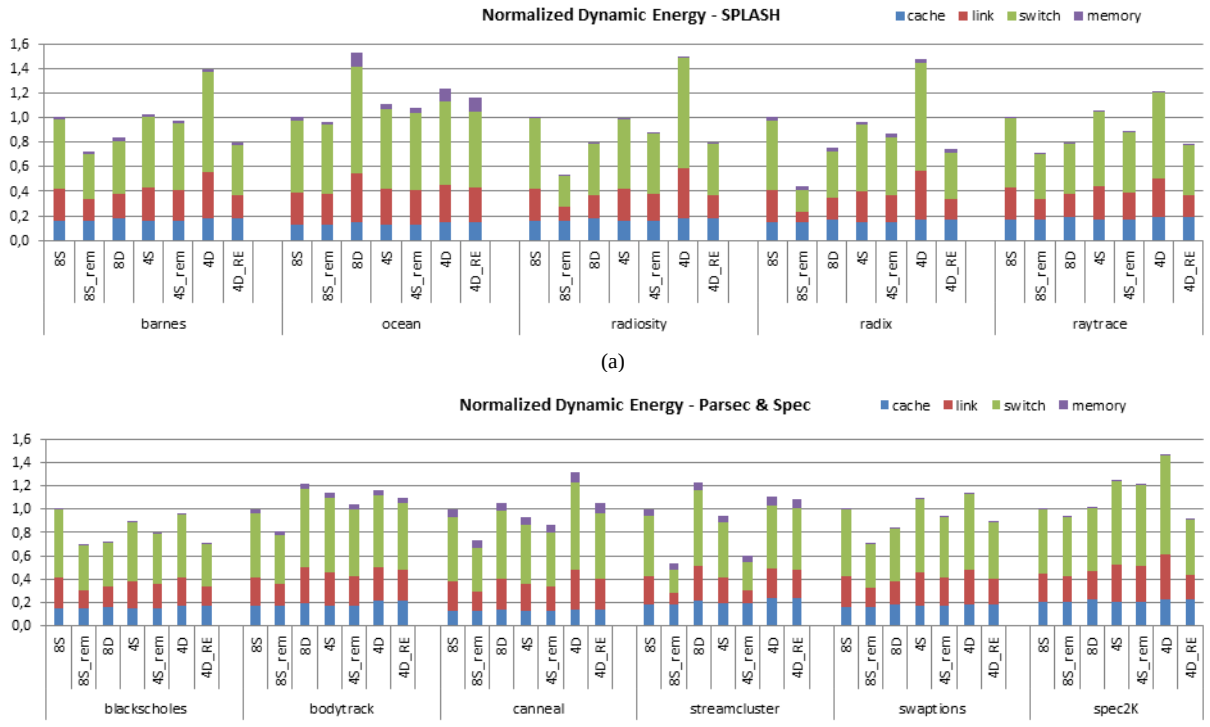
4S_rem solution is not as effective as 8S_rem in reducing dynamic energy, as the algorithm is less efficient in remapping blocks, and thus in reducing network traffic.

Dynamic energy consumption for DNUCA solutions is highly sensitive to the layout and the behavior of the applications. The 8D configuration usually is more energy efficient than 8S but less than 8S_rem. Also, when the migration mechanism is particularly efficient in reducing the access latency (for instance in Barnes and Bodytrack, Figure 13a and b), the increased control traffic due to search overwhelms the reduced data traffic (Figure 23 a and b). The net effect, in terms of energy, is an increase of the overall dissipated dynamic energy (Figure 25a and b). Exceptions are the executions of Ocean, Streamcluster and Bodytrack, for which the 8D presents the highest dynamic energy consumptions. In Ocean_8D, because only 60% of the hits are in the fastest way (Figure 13) due to the limited effect of migration, so that data traffic is similar to the 8S case, but the control traffic is much higher. In Ocean_4D and Ocean_4D_RE the migration is instead more efficient, and this produces a lower energy consumption. In Streamcluster and Bodytrack the distribution of accesses to the lines (Figure 15) for 8S already favors a 8P configurations, so that the increased control traffic on 8D is not compensated by the reduction of data traffic. For the other applications, 4D is the hungriest solution for dynamic energy due to ping-pong effects. Then, 4D_RE always exposes less dynamic energy than 4D, which comparable or lower to the energy consumption of 8D. In particular, it is lower than 8D for Ocean, Streamcluster, and Bodytrack, for which the migration in 8D is less efficient than for other applications.

Figure 26a and 26b summarize the overall energy consumption, highlighting the static (split in cache and switch quotes) and dynamic contributions. The first observation is that the static component overwhelms the dynamic, so that its figure essentially determines the overall energy consumption. As static energy is strictly related to execution time, the overall energy consumption has a similar trend as execution time and, in general, also if some faster configurations expose a higher dynamic energy consumption, this is more than compensated for by the reduced static energy due to shorter execution times. As a consequence, **8S_rem, 8D and 4D_RE solutions induce the lowest overall energy consumption** and the energy saving can be substantial: up to 18% for Barnes and Raytrace (8D and 4D_RE), almost similar values for Canneal (8S_rem, 8D and 4D_RE), up to 20% for Radix (8S_rem) and Blackscholes (8D and 4D_RE). Ocean conversely exposes an increased energy consumption for all DNUCA solutions, as they suffer of performance degradation. **4D solutions are more energy hungry when the ping-pong effects determine a performance degradation** (Radiosity, Radix, SPEC2K); in these cases, the energy demand is increased by the increased dynamic energy consumption.

Difference in power consumption between 8S and 4S is of about 10% or more only for Ocean and SPEC2K, due to the miss increase from 8S to 4S in Ocean and to the distribution of accesses in SPEC2K. For the other applications the difference is smaller, and typically originating from the different distribution of accesses. Lastly, **in the case of 4S_rem, the same considerations made on performance can be done**.

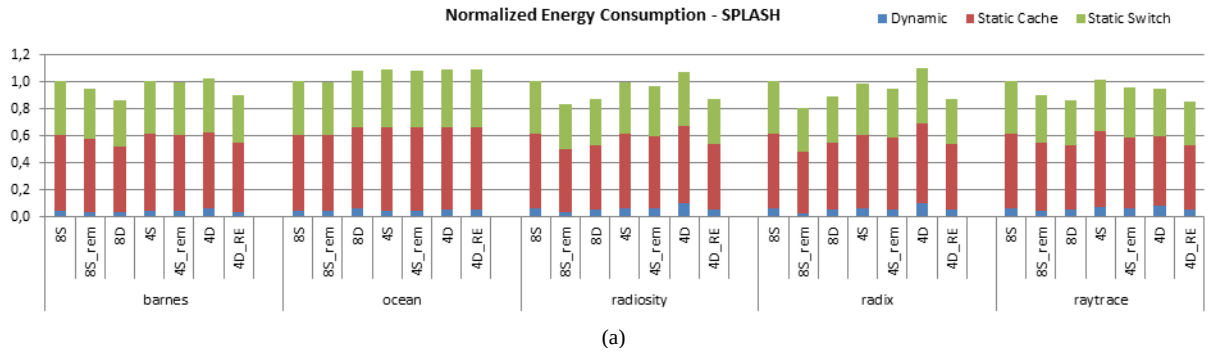
To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier



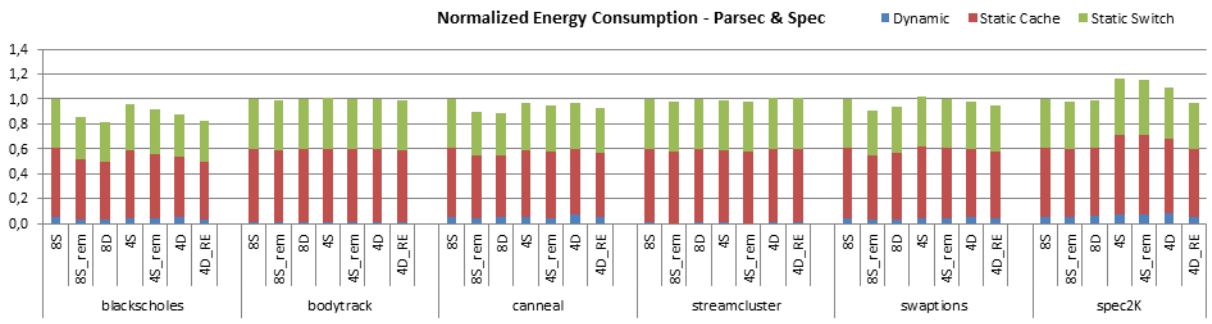
(a)

(b)

Figure 25: Normalized Dynamic Energy Consumption for the SPLASH (a) and Parsec & Spec (b) applications.



(a)



(b)

Figure 26: Normalized Energy Consumption for the SPLASH (a) and Parsec & Spec (b) applications.

doi: 10.1016/j.future.2017.06.001. © Elsevier

6.3 Portability of results

In the previous sections, we have considered an 8-core reference system. Then, in the case of memory layout optimization, we gathered bank usage statistics from the execution of each application with a certain input, and then evaluating the optimized memory mapping running again the application with the same input. To confirm the validity of our results, we must consider that a) the number of cores integrated on a chip can increase over time, and b) the proposed memory layout optimizations need to provide performance/energy improvements for a remapping table generated with an assigned input set, also when the application is run with different input sets. Similarly, also a partial execution on one input should be enough to optimize a whole execution. In this section, we address and discuss these issues.

Figure 27 presents the Normalized Execution Time for the SPLASH (a) and Parsec & Spec benchmark in the case of 16 cores. Results confirm that the considerations given for 8 cores still hold for 16 cores. We do not extend further our analysis as, for a higher number of cores devoted to the execution of mainstream applications [63], hierarchical solutions would be preferable to limit the storage and traffic costs associated with hardware coherence [61], [62]. In such hierarchical schemes, there are clusters of cores, each sharing a L2 cache, while L3 (or higher) cache levels interconnect the L2 caches. The architecture of each cluster is indeed like the ones analyzed in this paper, so our conclusions can be applied to the design of such clusters.

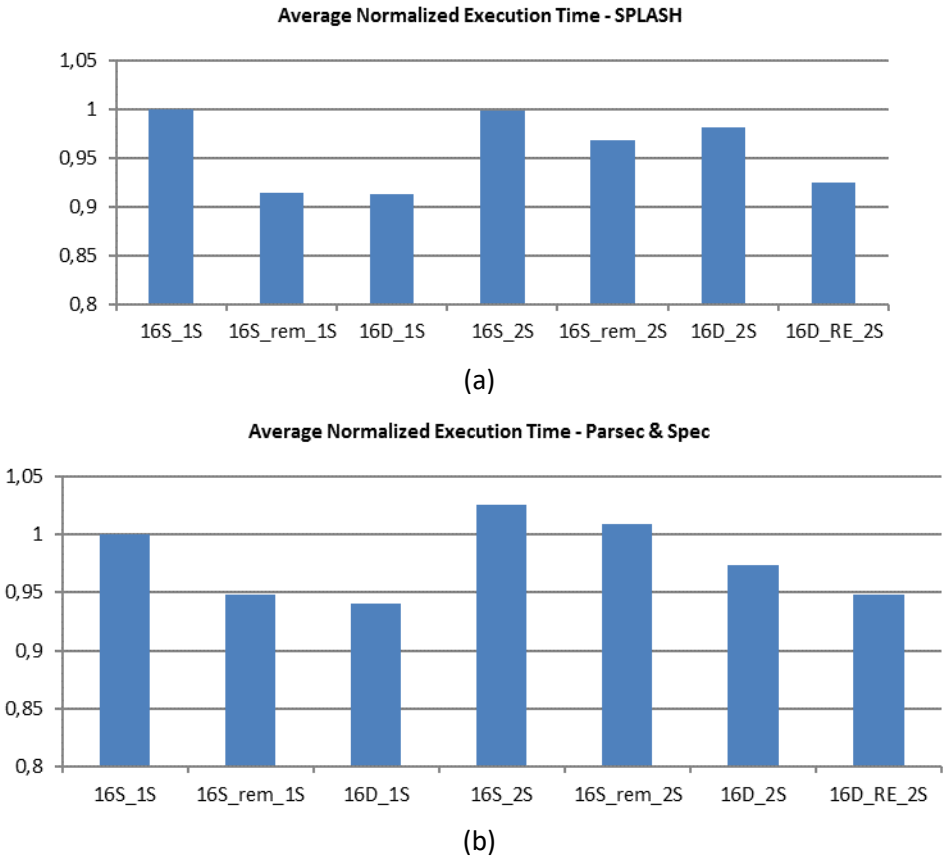


Figure 27. Average Normalized Execution time for the SPLASH (a) and Parsec & Spec (b) benchmarks, in the case of 16 cores, by considering all the layout and policies analyzed in the paper. 1S represents the one-side layout, 2S represents the two-sides layout.

As for the practical implementation of the memory layout optimizations, we have performed two experiments in the 8P layout, namely i) 8S_rem-short, where we have determined the remapping tables with an 800M instruction run, and then we have applied such mapping to the whole execution of the application; and ii) 8S_rem_other, where we have evaluated the remapped application with a different input set (in particular, we used the simsmall input sets [64][65][66]), and then we have applied such mapping to the whole execution of the application. Results in Figure 28 indicate that, once a remapping table is obtained, such a table is robust enough to be successfully applied to

doi: 10.1016/j.future.2017.06.001. © Elsevier

executions using different input sets, and that a shorter execution (e.g. the first part) can be enough to fruitfully optimize the whole run of the applications.

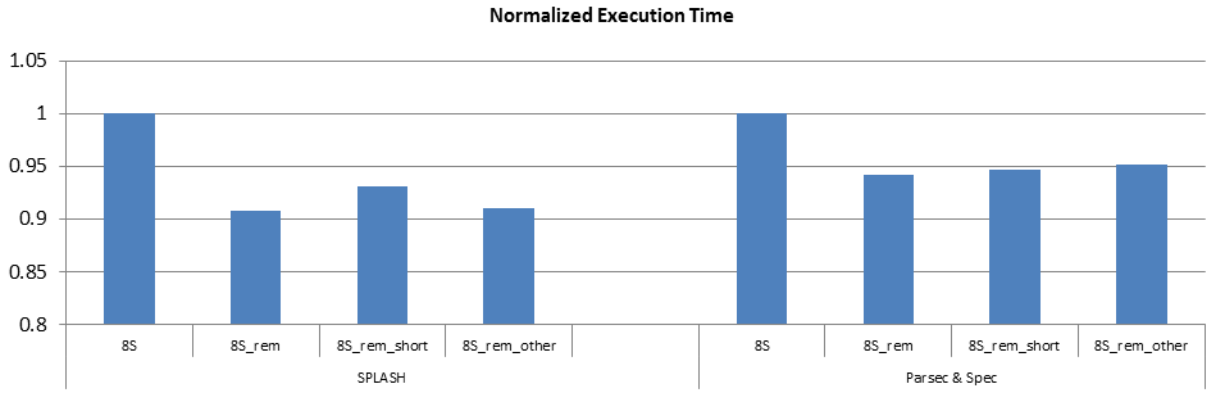


Figure 28. Normalized Execution Time for the SPLASH and Parsec&Spec Benchmark when utilizing, for populating the remap table, the same run length with the same input (8S_rem), a shorter Run (8S_rem_short) and a different input set (8S_rem_other).

7 Related works

NUCA caches have been proposed for monoprocessor systems in [5]. In this paper, the concepts of Static (SNUCA) and Dynamic NUCA (DNUCA) were introduced, and it was shown that NUCAs outperform conventional monolithic caches (UCA) when wire delay effects are significant, even when UCA caches are internally split in arrays to optimize power consumptions or performance [5], [16]. At the cost of increased network traffic and more sophisticated data management techniques, DNUCAs outperform also SNUCAs, when evaluated in mono-processor systems. Various works have then extended the monoprocessor design in order to optimize performance or energy efficiency [13],[28], [30],[31].

With the advent of CMPs, the NUCA concept has been extended to systems that adopt a large, shared, last-level-cache, to satisfy bandwidth requirements of CMPs [6]. The main issue is to exploit the wide variance of access times of large multi-banked caches, depending on the placement of data, to improve performance [52].

Beckmann and Wood [2], compare SNUCA and DNUCA designs in a 8-cpu CMP system where the shared L2 NUCA cache resides in the middle of the chip and is surrounded by the cores (2 by each side of the cache). They highlighted that the block migration mechanism of DNUCA is less effective in their CMP systems with respect to a single core system as 40-60% of the hits in the case of commercial and scientific workloads are in the central ways, due to conflicting accesses to shared blocks. In the DNUCA they adopt a two-phase search mechanism and show that the search mechanism policy impacts DNUCA cache performance.

Huh et al. [6] compare SNUCA and DNUCA performance in a CMP system in which 16 processors are connected to a 16 MB NUCA cache in a two-sides layout in line with Figure 1, finding that, on many of the considered applications, there is little gain for DNUCA over SNUCA, due to conflict hits. To limit useless migrations induced by conflict hit on shared data, they adopt saturated counters embedded in cache tag, and they introduce the concept of smart search to limit search related traffic. Foglia et al. [12] highlight that performance of SNUCA caches in the layouts of Figure 1 are strictly tied to the distribution of accesses to the cache. To reduce further the hit-latency of both SNUCA and DNUCA schemes, solution based on replication and ad-hoc data placement have been introduced.

In a two sides layout, Re-NUCA [40] introduces the replication of shared blocks accessed by processors located at the opposite sides of the shared cache, each of them migrating towards the side of the requesting core, as for private blocks.

In a SNUCA, Victim Replication (VR) [3] adds the replication of victims of L1 cache in LLC local banks. Local replication reduces hit latency but can increase LLC misses. Adaptive Selective Replication [8] dynamically monitors workload behaviors, and replicates read-only cached copies only when it is estimated that the benefit of replication (lower LLC hit latency) exceeds the cost (more LLC misses). In ESP NUCA [47] a data is initially placed in a bank local to the requesting core, then, if accessed by a different core, it is placed in the location based on the shared mapping (i.e. on a standard SNUCA mapping based on interleaving addressing), and replicated in other local banks, if needed.

doi: 10.1016/j.future.2017.06.001. © Elsevier

Reactive NUCA [48] deals with data placement and replication at memory page level, guided by the OS. The solution [53], instead, places data according to SNUCA scheme, and then it replicates data in local banks, according to locality info derived from the usage of blocks. FP-NUCA [52] introduces a solution that, in a shared L2 NUCA cache, allocates blocks according to the concept of bankset similarly to a DNUCA, allowing migration of blocks along a bankset on hit. It introduces also a router architecture that gives priority to NUCA search messages. Such a solution, adapted to a dance-hall configuration like the two presented in Figure 2, is similar to our proposed DNUCA architecture in a 44P architecture, but it does not deal with ping-pong effects. Other proposals [26][50] try to reduce the cost of a search, as a basic parallel search in DNUCA introduces significant network traffic and additional dynamic power consumption, although it is one of the most performing schemes.

An organization of banksets similar to the one adopted in this paper, and the consequent migration mechanism, has been proposed in [59] for a tiled architecture. With such a solution, they achieve performance and power improvement with respect to a conventional tiled SNUCA cache. A simpler approach has been proposed in [44], which utilizes the number of accesses to a bank to i) migrate data, and ii) switch off the bank if this does not hurt performance. To reduce static energy consumption in tiled SNUCAs, Dani et al. [42] propose to remap a certain subset of LLC slices (i.e. the cache bank local to a core) onto the remaining ones, by utilizing a remap table. Such table is similar to the one proposed in our work, although it is utilized with a different objective. They try to leave some slices without accesses so that they can be switched off to save energy. To populate the table, in [41] they utilize offline genetic algorithms based on application profiling, while in [43] they propose a runtime technique that estimates the working set size and the consequent cache space need.

Many of the previous solutions aim at reducing the access latency of the NUCA cache utilizing hardware mechanisms, which in the end try to move the most accessed data towards low latency banks. A similar mechanism can be performed in software, at OS level or through remapping tables of memory addresses, populated using information obtained before execution or at run-time.

Cho et al. [49] propose an OS-assisted data allocation scheme based on page coloring. They adopt a data-to-cache bank mapping, so that consecutive memory blocks within a page can reside in the same cache bank. They use a first-touch color assignment, i.e., a page is assigned to a color that places it in the LLC slice nearest to the core that first touches that page. Awasthi et al. [45] introduce a mechanism that allows the hardware to dynamically change page color and page placement in the cache (i.e. a page can migrate from one bank to another). They define OS dynamic policies, evaluated periodically, that track cache usage and modulate the space (colors) assigned to the cores. They also recolor pages if the "optimal" home bank of a page is different than the one that was previously estimated. Performance is similar to a DNUCA in a configuration like Beckmann and Wood [2], but without the effort of implementing a typical DNUCA search mechanism. Also Chaudhuri [46] implements a page migration strategy, fully hardwired, based on the accesses history to each page, associated to each block. To migrate a page within LLC, the OS-assigned address (P) is converted into a different address (P'). Since the L1 cache and main memory continue to use the old mappings, tables are required at each core and at each L2 bank to perform translations back and forth between P and P', differently from [45]. The L2 remap tables are similar to ours but, in our case, we need to keep only one table entry per NUCA bank.

A complete software oriented approach for the management of NUCA caches has been proposed in Jin and Cho [55]. It is based on profile and ties with previous works on CC-NUMA [58]. They employ compiler hints to guide the OS in selecting a near-optimal page coloring. At compile time, the program is profiled and access counts per page and per core are collected. Each page is mapped to a tile that is expected to generate the highest number of accesses to such page. Pages marked as being shared equally by all cores are distributed at cache line granularity instead of page size granularity. [39] was an initial investigation from the authors of the present work, to understand the effectiveness of optimizing SNUCA architecture via profiling base memory remapping. A simplified memory model was utilized to estimate latency improvements, and the remapping was implemented at virtual page level, highlighting the dependency of the mapping on the hardware architecture. Moreover, such a study did not investigate the performance trade-off among DNUCA and SNUCA in the two layouts of Figure 1.

As a summary, the design space of NUCA caches has been deeply explored concerning SNUCA vs DNUCA performance comparison [6], [2], performance optimizations based on replication [3], [8], [53] and to limit the effect of ping-pong [40], optimal private-shared partitioning [47], improvements in the search [26] also via replication [50], optimal placement of copies to blocks - hardwired [59], [46] or with OS support [48], [49], [45], exploiting also hints derived at compile time via a profiling phase [55]. These studies assume a fixed layout (i.e.: cores all placed at one side of the NUCA or half and half at the two opposite sides of it), and do not give an evaluation of which are the combined effects on performance by varying layouts and data management policies together (such as migration, replication and data mapping). Conversely the main focus of this paper is the comparison of the one-side

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

and the two-side layouts (Figure 1), taking into consideration the basic implications of the two designs, i.e. conflict hits and accesses distribution, and how the management of these issues can impact performance and power consumption. In this analysis we explore, in an integrated fashion, the effects of the migration and replication mechanisms in DNUCA and the possibility of memory layout restructuring in SNUCA, which are the main classes of approaches adopted to cope with the weaknesses that arise, respectively, in DNUCA and SNUCA designs for multi-core processors.

8 Conclusions

NUCA design is gaining momentum to build efficient chip-multiprocessor last-level caches. In fact, it enables the keeping up with the strong need of on-chip capacity and bandwidth increase thanks to a modular organization built around an ad-hoc interconnection. Several strategies have been proposed to exploit the non-uniform access time of NUCA caches to improve performance and/or reduce power consumption. These techniques can act both at hardware and software level but they never consider the combined effects of such strategies with the physical layout, i.e. the position of cores against the last-level cache banks. Nevertheless, different physical layouts have been utilized to implement high performance multicore systems.

In this work, we investigated the performance and energy effects of two basic layouts for building NUCA systems, the first with all cores connected at the same side of the shared NUCA cache (8P), the second with the cores evenly split between two opposite sides of it (44P). To highlight properly the different behavior of the two layouts, SNUCA and DNUCA organizations have been considered in integration with the main classes of hardware/software optimizations suitable in each design and layout.

Experimental results on benchmarks from Parsec and Splash suites showed that 8P layout exposes higher capability both to attract data towards faster ways in DNUCA, and more effective optimization space for software remapping strategies in SNUCA. Consequently, DNUCA 8P and software-remapped SNUCA made it possible to achieve the best performance and lowest power consumption. Furthermore, we showed that remapping-based solutions allow to minimize on-chip network usage thanks to locality exploitation and to the lack of the cache-wide incremental/broadcast search cost, typical of DNUCA. Then, if 44P layout is mandatory due to external constraints, our results highlighted that performance similar to 8P layout can be achieved only by introducing a replication scheme, which requires additional complexity in the coherence protocol. We showed the portability of these guidelines by both increasing the number of cores, and across different application profiles.

9 Acknowledgements

The authors want to thank Windriver Ltd for granting us the Simics software and licenses as well as the anonymous reviewers for the insightful comments that helped us to improve the paper.

10 References

[1] K. Olukotun, B.A. Nayfeh, et al., “The case for a single-chip Multiprocessor” *Proc. of the 7th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, pp. 241-251, Oct. 1996.

[2] B.M. Beckmann and D.A. Wood, “Managing Wire Delay in Large Chip-Multiprocessor Caches” *Proc. of the 37th annual IEEE/ACM Int. Symp. on Microarchitecture*, Portland, OR, USA, pp. 319-330, Dec. 2004.

[3] M. Zhang and K. Asanovic, “Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors” *Proc. of the 32nd annual Int.Symp. on Computer Architecture*, Madison, WI, USA, pp. 336-345, June 2005.

[4] R. Ho, K.W. Mai and M.A. Horowitz, “The future of wires” *Proc. of the IEEE*, 89(4), pp. 490-504, April 2001.

[5] C. Kim, D. Burger and S. W. Keckler, “An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches” *Proc. of the 10th Int.Conf. on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, USA, pp. 211-222, Oct 2002.

[6] Jaehyuk Huh; Changkyu Kim; Shafi, H.; Lixin Zhang; Burger, D.; Keckler, S.W., "A NUCA Substrate for Flexible CMP Cache Sharing," in Parallel and Distributed Systems, IEEE Transactions on, vol.18, no.8, pp.1028-1040, Aug. 2007.

[7] J. Duato, S. Yalamanchili and L. Ni, *Interconnection Networks an Engineering Approach*. San Francisco, CA. Morgan Kauffmann, Elsevier, 2003.

[8] B. M. Beckmann, M. R. Marty, D. A. Wood, "ASR: Adaptive Selective Replication for CMP Caches" *Proc. of the 39th annual IEEE/ACM Int. Symp. on Microarchitecture*, Orlando,FL, pp. 443-454, Dec. 2006.

[9] C. Bienia, et al., “The PARSEC Benchmark Suite: Characterization and Architectural Implications” *Proc. of the 17th Int. Conf. on Parallel Architectures and Compilation Techniques*. Toronto, Canada, pp. 72-81, Oct. 2008.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

[10] K. Gharachorloo, M. Sharma, S. Steely and S. Van Doren, "Architecture and Design of AlphaServer GS320" *Proc. of the 9th Int.Conf. on Architectural Support for Programming Languages and Operating Systems*, Cambridge, MA, USA, pp. 13-24, Nov. 2000.
[11] P. Foglia, F. Panicucci, C. A. Prete, M. Solinas, Analysis of Performance dependencies in NUCA-based CMP systems. 21st Intern. Symp. on Computer Architecture and High Performance Computing, Oct. 28 - 31, Sao Paulo, Brazil, pp. 49-56, 2009.
[12] P. Foglia, F. Panicucci, C. A. Prete, M. Solinas, An Evaluation of Behaviors of S-NUCA CMPs running scientific workload. 12th EUROMICRO Conference on Digital System Design, Architectures, Methods and Tools (DSD09), August 27-29, Patras, Greece, pp. 26-33, 2009.
[13] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli and C.A. Prete, "A power-efficient migration mechanism for D-NUCA caches". In *Proc. of the Design, Automation and Test in Europe09*, Nice, France, pp. 598-601, Apr. 2009.
[14] Wind River Simics, http://www.windriver.com/products/simics/.
[15] Winsconsin Multifacet GEMS Simulator, http://www.cs.wisc.edu/gems/
[16] Muralimanohar, N., Rajeev B., and Norman P. Jouppi."CACTI 6.0: A tool to model large caches." HP Laboratories (2009).
[17] Predictive Technology Model (PTM), http://www.eas.asu.edu/~ptm/
[18] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., Gupta, A., "The SPLASH-2 programs: characterization and methodological considerations". *Proc. of the 22th Int. Symp. on Computer Architecture*, S. Margherita Ligure, Italy, pp.24-36, June 1995
[19] http://techreport.com/review/21987/intel-core-i7-3960x-processor
[20] http://techreport.com/review/18581/intel-core-i7-980x-extreme-processor
[21] http://techreport.com/review/20188/intel-sandy-bridge-core-processors
[22] http://www.anandtech.com/show/2978/amd-s-12-core-magny-cours-opteron-6174-vs-intel-s-6-core-xeon/14
[23] B. Sinharoy et al: IBM Power7 Multicore Processor, IBM J. RES. & DEV. VOL. 55 NO. 3 PAPER 1, MAY/JUNE 2011.
[24] Sinharoy, B., et al. IBM POWER8 processor core microarchitecture. IBM Journal of Research and Development 59.1 (2015): 2-1.
[25] Stuecheli, Jeff. "POWER8."Hot Chips. Vol. 25. 2013.
[26] Lira, Javier, Carlos Molina, and Antonio González. "Hk-nuca: Boosting data searches in dynamic non-uniform cache architectures for chip multiprocessors." Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International.
[27] Lira, Javier, et al. "Replacement techniques for dynamic NUCA cache designs on CMPs."The Journal of Supercomputing64.2 (2013): 548-579.
[28] P. Foglia, et al, "A Workload Independent Energy Reduction Strategy for D-NUCA Caches, Journal of Supercomputing, 10.1007/s11227-013-1033-5, October 2013.
[29] A.Bardine, P. Foglia, F. Panicucci, J. Sahuquillo, M. Solinas, "Energy Behaviors of NUCA caches in CMPs". 14th EUROMICRO Conf. on Digital System Design, OULU, Finland, pp: 746-753, Aug 31 -Sept. 2, 2011.
[30] Z. Chisti, M.D. Powell, T.N. Vijaykumar: "Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures" 36th Int. Symp. On Microarchitecture, pp.55-66, Dec. 2003.
[31] A. Bardine, M. Comparetti, P. Foglia, C. A. Prete, Evaluation of Leakage Reduction Alternatives for Deep Submicron Dynamic Nonuniform Cache Architecture Caches, IEEE Transactions on Very Large Scale Integration Systems, vol.22(1), pp.185-190, Jan. 2014.
[32] K. Strandberg. "Which OS? Considerations for Performance-asymmetric, Multi-core Platforms," Research at Intel, White Paper.
[33] CACTI "An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model", HP Laboratories, http://www.hpl.hp.com/research/cacti/, version 6.5 software download http://www.hpl.hp.com/research/cacti/cacti65.tgz.
[34] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. 2009. ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration. In Proceedings of the Conference on Design, Automation and Test in Europe (DATE '09). European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 423-428.
[35] A. B. Kahng, B. Lin and S. Nath, "Explicit Modeling of Control and Data for Improved NoC Router Estimation", Proc. DAC, 2012, pp. 392-397.
[36] Barrow-Williams, Nick, Christian Fensch, and Simon Moore. "A communication characterisation of Splash-2 and Parsec." Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on. IEEE, 2009.
[37] S. Bartolini, P. Foglia, C. A. Prete, M. Solinas, Coherence in the CMP ERA: Lesson learned in designing a LLC architecture, WSEAS Transactions on Computers, Vol. 13, pp. 195-207, March 2014.
[38] S. Bartolini and C. A. Prete. 2005. Optimizing instruction cache performance of embedded systems. ACM Trans. Embed. Comput. Syst.4, 4 (November 2005), 934-965.
[39] S. Bartolini, P. Foglia, M. Solinas, and C. A. Prete. 2010. Feedback-Driven Restructuring of Multi-Threaded Applications for NUCA Cache Performance in CMPs. In Proceedings of the 2010 22nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '10). IEEE Computer Society, Washington, DC, USA, 87-94.
[40] P. Foglia, M. Solinas , Exploiting Replication to Improve Performances of NUCA-Based CMP Systems, ACM Transactions on Embedded Computing Systems,doi: 10.1145/2566568, Vol. 13 Issue 3s, Art. N. 117, March 2014.

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier

[41] A. M. Dani, B. Amrutur, and Y. N. Srikant. 2011. Applying genetic algorithms to optimize the power in tiled SNUCA chip multicore architectures. In Proceedings of the 2011 ACM Symposium on Applied Computing (SAC '11). ACM, New York, NY, USA, 1090-1091.
[42] Mandke, Aparna, et al. "Optimizing Power in Tiled S-NUCA CMP Architectures Using Remap Table." (2010). IISc-CSA-TR-2010-1.
[43] Dani, A.M.; Amrutur, B.; Srikant, Y.N., "Adaptive Power Optimization of On-chip SNUCA Cache on Tiled Chip Multicore Architecture Using Remap Policy," in Architecture and Multi-Core Applications (WAMCA), 2011 Second Workshop on, vol., no., pp.12-17, 26-27 Oct. 2011.
[44] H. K. Kapoor, S. Das, and S. Chakraborty. 2015. Static energy reduction by performance linked cache capacity management in tiled CMPs. In Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC '15). ACM, New York, NY, USA, 1913-1918.
[45] Awasthi, M., Sudan, K., Balasubramonian, R. and Carter, J.B. 2009. Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In proc. 15th Int. Symp. on High Performance Computer Architecture (HPCA 2009), Raleigh, North Carolina, Feb. 14-18.
[46] Chaudhuri, M. 2009. PageNUCA: Selected policies for page-grain locality management in large shared chip-multiprocessor caches. *In Proceedings of the 15th IEEE International on High Performance Computer Architecture* (HPCA 2009), Raleigh, North Carolina, Feb. 14-18.
[47] Merino, J., Puente, V. and Gregorio, J.A. 2010. ESP-NUCA: A Low-cost Adaptive Non-Uniform Cache Architecture. In Proc. 16th IEEE Intern. Symposium on High-Performance Computer Architecture, Bangalore, India, pp. 1-10.
[48] Hardavellas, N., Ferdman, M., Falsafi, B. and Anastasia Ailamaki. 2009. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th annual international symposium on Computer architecture* (ISCA '09). ACM, New York, NY, USA, 184-195.
[49] Cho, S. and Jin, L. 2006. Managing Distributed, Shared L2 Caches through OS-Level Page Allocation. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (MICRO 39). IEEE Computer Society, Washington, DC, USA, 455-468.
[50] N. Chaturvedi, A. Subramaniyan, and S. Gurunarayanan. 2015. An adaptive migration---replication scheme (AMR) for shared cache in chip multiprocessors.*J. Supercomput.*71, 10 (October 2015), 3904-3933.
[51] Y. Jin, E. J. Kim, K. H. Yum. A Domain-Specific On-Chip Network Design for Large Scale Cache Systems.In IEEE 13th International Symposium on High Performance Computer Architecture, 2007., pp.318-327, 10-14 Feb. 2007.
[52] A. Arora, M. Harne, H. Sultan, A. Bagaria and S. R. Sarangi, "FP-NUCA: A Fast NOC Layer for Implementing Large NUCA Caches," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 9, pp. 2465-2478, Sept. 1 2015.
[53] Hijaz, Farrukh, et al. "Locality-aware data replication in the last-level cache for large scale multicores. "The Journal of Supercomputing", 72.2 (2016): 718-752.
[54] S. Bartolini and C. A. Prete. A proposal for input-sensitivity analysis of profile-driven optimizations on embedded applications. In *Proc. of the 2003 workshop on MEmory performance: DEaling with Applications, systems and architecture* (MEDEA '03). 2003. ACM, New York, NY, USA, 70-77.
[55] L. Jin and S. Cho. SOS: A Software-Oriented Distributed Shared Cache Management Approach for Chip Multiprocessors. In Proceedings of PACT, 2009.
[56] L. L. Pilla, C. P. Ribeiro, P. Coucheney, et al., "A topology-aware load balancing algorithm for clustered hierarchical multi-core machines", in Future Generation Computer Systems, Volume 30, January 2014, Pages 191-201, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2013.06.023.
[57] R. Sivaramakrishnan and S. Jairath, "Next Generation SPARC Processor Cache Hierarchy," Hot Chips 26, 2014.
[58] J. Marathe and F. Mueller. "Hardware profile-guided automatic page placement for ccNUMA systems," Proc. PPoPP, March 2006.
[59] S. Das and H. K. Kapoor, "Exploration of Migration and Replacement Policies for Dynamic NUCA over Tiled CMPs," VLSI Design (VLSID), 2015 28th International Conference on, Bangalore, 2015, pp. 141-146.
[60] P. Lotfi-Kamran, B. Grot and B. Falsafi, "NOC-Out: Microarchitecting a Scale-Out Processor," *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on*, Vancouver, BC, 2012, pp. 177-187.
[61] Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin. 2012. Why on-chip cache coherence is here to stay. Commun. ACM 55, 7 (July 2012), 78-89.
[62] A. Ros, M. Davari and S. Kaxiras, "Hierarchical private/shared classification: The key to simple and efficient coherence for clustered cache hierarchies,"2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, 2015, pp. 186-197.
[63] D. Johnson, M. Johnson, J. Kelm, W. Tuohy, S. Lumetta and S. Patel, "Rigel: A 1,024-Core Single-Chip Accelerator Architecture," in IEEE Micro, vol. 31, no. 4, pp. 30-41, July-Aug. 2011.doi: 10.1109/MM.2011.40.
[64] A Memo on Exploration of SPLASH-2 Input Sets PARSEC Group Princeton University, June 2011, available at http://parsec.cs.princeton.edu/doc/memo-splash2x-input.pdf
[65] The Parsec and SPLASH-2x benchmark suite, available at http://parsec.cs.princeton.edu/parsec3-doc.htm
[66] Christian Bienia and Kai Li, Fidelity and Scaling of the PARSEC Benchmark Inputs. In Proceedings of the IEEE International Symposium on Workload Characterization, December 2010

doi: 10.1016/j.future.2017.06.001. © Elsevier

[67] E. F. Y. Young, C. C. N. Chu and M. L. Ho, "Placement constraints in floorplan design," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 12, no. 7, pp. 735-745, July 2004.
[68] Matteo Monchiero, Ramon Canal, and Antonio González. Design space exploration for multicore architectures: a power/performance/thermal view. In Proceedings of the 20th annual international conference on Supercomputing, ACM, New York, NY, USA, 177-186, 2006.
[69] Jordi Cortadella, Javier de San Pedro, Nikita Nikitin, and Jordi Petit. Physical-aware system-level design for tiled hierarchical chip multiprocessors. In Proceedings of the 2013 ACM international symposium on International symposium on physical design (ISPD '13), ACM, New York, NY, USA, 3-10, 2013.
[70] Chen, Xi, Jiang Hu, and Ning Xu. "Regularity-constrained floorplanning for multi-core processors." Integration, the VLSI Journal 47.1 (2014): 86-95.
[71] C. Bienia, S. Kumar and Kai Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors," 2008 IEEE International Symposium on Workload Characterization, Seattle, WA, 2008, pp. 47-56.
[72] L. Chen, Y. Wang, et al., “Multiple-combinational-channel: A network architecture for workload balance and deadlock free”, in Future Generation Computer Systems, Volume 56, March 2016, Pages 238-246, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2015.08.013.
[73] J. L. Henning, "SPEC CPU2000: measuring CPU performance in the New Millennium," in Computer, vol. 33, no. 7, pp. 28-35, Jul 2000.
[74] D. Mulnix, “Intel® Xeon® Processor E7-8800/4800 v4 Product FamilyTechnical Overview”, available at https://software.intel.com/en-us/articles/intel-xeon-processor-e7-8800-4800-v4-product-family-technical-overview
[75] A. Jaleel, J. Nuzman, A. Moga, S. C. Steely and J. Emer, "High performing cache hierarchies for server workloads: Relaxing inclusion to capture the latency benefits of exclusive caches," 2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA), Burlingame, CA, 2015, pp. 343-353.
[76] “Find the Right Intel® Xeon® Processor-Based Server to Match Your Workload”, Solution Brief Available at http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/find-the-right-xeon-processor.pdf

To appear in Elsevier Future Generation Computer Systems, Before proof version.
doi: 10.1016/j.future.2017.06.001. © Elsevier