Re-NUCA: Boosting CMP performance through block replication

Pierfrancesco Foglia, Cosimo Antonio Prete, Marco Solinas
Dipartimento di Ingegneria dell'Informazione
Università di Pisa
Via Diotisalvi, 2 – 56100 Pisa (PI) Italy
{foglia, prete, marco.solinas}@iet.unipi.it

Giovanna Monni
IMT Institute for Advanced Studies Lucca
Piazza S. Ponziano, 6 55100 Lucca
giovanna.monni@imtlucca.it

All the authors are members of *HiPEAC* – European Network of Excellence

Abstract— Chip Multiprocessor (CMP) systems have become the reference architecture for designing micro-processors, thanks to the improvements in semiconductor nanotechnology that have continuously provided a crescent number of faster and smaller per-chip transistors. The interests for CMPs grew up since classical techniques for boosting performance, e.g. the increase of clock frequency and the amount of work performed at each clock cycle, can no longer deliver to significant improvement due to energy constrains and wire delay effects. CMP systems generally adopt a large last-level-cache (LLC) (typically, L2 or L3) shared among all cores, and private L1 caches. As the miss resolution time for private caches depends on the response time of the LLC, which is wire-delay dominated, performance are affected by wire delay. NUCA caches have been proposed for single and multi core systems as a mechanism for tolerating wire-delay effects on the overall performance.

In this paper, we introduce a novel NUCA architecture, called Re-NUCA, specifically suited for (but not limited to) CMPs in which cores are placed at different sides of the shared cache. The idea is to allow shared blocks to be replicated inside the shared cache, in order to avoid the limitations to performance improvements that arise in classical D-NUCA caches due to the conflict hit problem.

Our results show that Re-NUCA outperforms D-NUCA of more then 5% on average, but for those applications that strongly suffer from the conflict hit problem we observe performance improvements up to 15%.

Keywords-component; cache memory, NUCA, block replication, CMP systems

I. INTRODUCTION

Chip Multiprocessor systems (CMPs) [1,2] have become the reference architecture for improving performance of modern and future microprocessor systems. In a typical design, each core has its own private caches, while the Last Level Cache (LLC) can be either private [3,4] or shared [5,6,7,8]; hybrid designs have been also proposed [9,10,11]. Shared caches usually minimize off-chip misses, but exhibit high access latencies since many requests must traverse long paths to reach distant cache banks; in this context, the *wire delay* [12] has a negative impact on the hit latency. For this reason, a sub-banked organization of the shared LLC, characterized by a non–uniform access time, would help in hiding wire delay negative effects on performance.

By exploiting this sub-banked cache organization, Non-Uniform Cache Access (NUCA) architecture has been proposed [13,14]: a NUCA is a bank-partitioned cache in which the banks are connected by means of a scalable communication infrastructure (typically, a Network-on-Chip, NoC [15,16]), and it is characterized by a non-uniform access time. Depending on the mapping policy, NUCA caches can be classified as *Static* (S-NUCA, with a fixed mapping between each physical address and a single bank) and *Dynamic* NUCA (D-NUCA, characterized by a dynamic mapping between each address and a set of banks, called bankset). In particular, in a D-NUCA blocks are able to migrate among the banks belonging to their bankset. This migration mechanism allows data to move towards the most frequently referring CPUs, with the purpose of reducing the average cache latency by storing the most frequently accessed blocks in banks close to the referring cpus (i.e. in the *faster way*).

In CMP configurations in which processors are placed at different sides of the shared D-NUCA cache, such as our reference system shown in Figure 1, performance improvements due to the migration process can be limited by the *conflict hit phenomenon* [23]. In CMP systems, such problem is related to the presence of shared blocks that are accessed by threads running on processors placed at opposite sides of the shared cache. In particular, such blocks migrate each time a request coming from an L1 cache hits the block in a D-NUCA bank, in the direction of the faster way of the L1 requestor. If such L1s stay at opposite D-NUCA sides (e.g. L1-0 and L1-4 in Figure 1), cached blocks alternatively migrate up and down in the pertaining banksets, and can't be stored in any of the faster ways. Figure 2 shows the conflict hit phenomenon for a shared block accessed by P0 and P1. Due to the conflict hit phenomenon, improvements on average D-NUCA response time are limited for shared blocks accessed by opposite cpus.

In this paper, we introduce the Re-NUCA, a novel cache architecture that allows *limited replication* for shared blocks accessed by processors placed at opposite chip sides. In particular, our solution lets at most two independent copies of the same block to be stored in the same shared cache, each of them migrating towards the closest cache side, named *target side*. Private blocks, as well as shared blocks that are accessed only by cpus at the same Re-NUCA side, are able to migrate as in a classical D-NUCA cache. Banks storing the two copies are able to recognize if a request comes from a cpu placed at the target side or at the opposite side. In this way, we are able to boost the overall system performance by

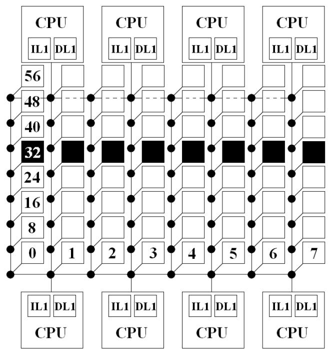

Figure 1. The considered CMP configuration. Small squares are the NUCA banks, black circles are NoC switches, white big squares represent cpu+L1 cache nodes. Filled squares represent the entry points for off-chip blocks (Collector). Bank columns are the banksets, while rows of the bank matrix are called ways, and are numbered from 1 (first bank: 0) to 8 (first bank: 56)

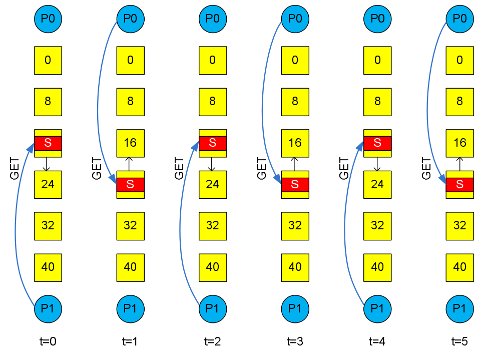

Figure 2. The conflict hit phenomenon. At t=0, P1 accesses a shared block S stored in bank 16, and the block migrates to bank 24. At t=1, P0 accesses the S block, that migrates back to bank 16. At subsequent time instants, both P1 and P0 alternatively access the block, so it keeps "pingpong-ing" between banks 16 and 24, and is not able to reach the faster ways with respect to none of the referring cpus

avoiding the *conflict hit* phenomenon, as only the closest bank that holds a copy of the shared block will provide the L1 requestor with the referred datum (that then migrates), while the farthest bank will behave as in the miss case.

Results show that, with respect to an optimized D-NUCA scheme, performance are boosted on average of more than 5%, but for those applications that strongly suffers from the conflict hit problem the speedup varies between 9% and 15%, with very little or no degradation in the L2 miss rate.

The rest of this paper is organized as follows. Section 2 discusses some related work. Section 3 describes the Re-NUCA replication protocol. Section 4 presents our evaluation methodology. Section 5 discusses our results. Section 6 concludes the paper and presents some extensions as future works.

II. Related Works

Different works have analyzed CMP systems that adopt a large last-level-cache characterized by a non-uniform access time. In order to exploit such feature, some have considered both S-NUCA and D-NUCA (migration based) strategies [7,14], none explicitly addressing the conflict hit problem; others have considered different replication policies [9,10,17,31], or both migration and replication [32].

Beckmann and Wood in [7] propose an 8-cpus CMP system in which each processor has its private L1 I&D caches and a huge shared L2 NUCA cache, both Static and Dynamic. They highlighted that block migration is less effective in CMP systems (with cpus distributed on more sides) as 40-60% of the hits in the case of commercial workloads (but also for some scientific applications) are in the central ways: this is due to shared blocks. They obtain significant performance improvements only adopting a smart search mechanism, but it's difficult to implement. Also Huh et al. in [14] propose a NUCA-based CMP system in which 16 processors share a 16 MB NUCA cache. They obtain improvements with D-NUCA respect to S-NUCA, but for those applications that exhibit a high sensitivity to the conflict hit problem the migration mechanism is less effective.

Other works do not explicitly deal with migration, instead they deal with private-shared cache partitioning and block replication. Chisthi et al. in [9] evaluate an hybrid design that tries to take advantage of both shared and private configuration for the last level cache, by proposing a set of techniques that act on data replication, communication between sharers and storage capacity allocation. However, such technique requires a non-scalable atomic bus for managing coherence. Cooperative Caching [10] exploits the benefits of both private and shared cache; the coherence protocol is a directory MOSI. It is based on a centralized structure, called CCE, which is not suitable for large-scale CMP systems; instead, Re-NUCA avoids the need of a centralized directory in order to increase scalability. The ASR mechanism [17] is proposed to optimize CMP performance by controlling replication of cache blocks, while ESP-NUCA [31] adopts a dynamic private-shared cache partitioning, with replication and victim management. However, both [17] and [31] manage coherence through the Token Coherence [33] protocol, and thus their scalability degree is limited by broadcast communication. Finally, Hardavellas et al. in [32] propose a tiled architecture in which the aggregation of all L2 tiles is seen as a shared NUCA, and adopt a hybrid hard-soft mechanism to control block placement, migration and replication among slices; however, this solution requires OS modification to support hard-soft interaction.

As a summary, for applications with a high degree of block sharing, [7] and [14] do not obtain a strong performance gain for the D-NUCA with respect to the S-NUCA, because the reference topology doesn't succeed in taking advantage from the blocks migration. Proposed replication schemes do not combine it with an efficient migration mechanism, or need hard-soft interaction to manage both aspects. Instead, in this paper we propose a solution that makes block migration effective in boosting performance while maintaining scalability, since the *limited replication* mechanism of Re-NUCA allows at most two L2 copies of a shared block without taking care of how many processors are sharing it.

III. RE-NUCA DESIGN

As the replication mechanism we introduce extends the behaviors of a classical D-NUCA, we considered the migration scheme with the FMA and Collector optimizations proposed in [29]. The architecture adopts a directory version of MESI as the baseline coherence protocol, in which the directory is *non-blocking* [19,20,21] and distributed. The cache hierarchy is inclusive, so there is no need for a centralized directory, as directory information is held in the TAG field of the L2 copy. The mapping policy between physical address and NUCA banksets is implemented using the low-order bits of the index field (interleaved mapping [22]). We consider a *per-hit* [13] block migration: a data promotion is triggered whenever a block request from an L1 results in a hit in any of the NUCA banks belonging to the block's bankset. As blocks can be stored in *any* of the bankset's banks, we assume, as the search policy, that L1-to-L2 requests are broadcasted to all the banks of the reference bankset.

In our implementation, we assume that both Read-Only and Read-Write blocks can be replicated. So, in the following, we consider that both Read requests (GET Shared, or GETS), and Read-With-Intent-To-Modify requests (GET eXclusive, or GETX) can be issued by any L1 cache.

A. D-NUCA basics: the FMA protocol

The D-NUCA migration protocol is adopted to dynamically move frequently accessed blocks among banks in order to be stored in faster ways. During the time interval in which a block is moving from the source to the destination bank, it is important to manage subsequent requests that could be issued by other L1 caches. A *false miss* [7,29] is a race condition that can arise when a subsequent request for a migrating block is received from the sender *after* the block has been issued (and the corresponding cache line has been deallocated), and by the receiver *before* the block has arrived. Such condition results in an extra off-chip access even if the referred block is actually cached.

To manage such race condition, the *False Miss Avoidance* (FMA) protocol has been proposed [29]. Such protocol eliminates the problem by ensuring that, at each time, at least one of the involved banks is aware of the fact that the block is cached. This is done by adopting an explicit *three-way* message exchange (START-ACK-END) between the sender and the receiver banks: in this way, the sender deallocates the cache line only when the acknowledgment from the receiver arrives, thus guaranteeing that at least one bank is always able to accept a subsequent request. Avoiding extra off-chip accesses is not just an efficiency problem, but also a correctness problem since there could be two (or more) unmanaged copies of the same block.

We adopted such efficient migration scheme in our solution since it is able to avoid the need of a centralized directory node [14] to track cached blocks and eliminate unnecessary extra off-chip accesses. Lazy migration [7] is no longer needed since the false miss problem is directly avoided by the migration process itself.

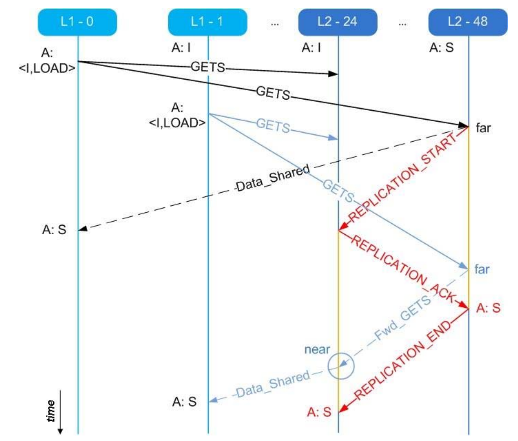

Figure 3. Sequence diagram representing the block replication. In case of hit in an L2 bank, the block is sent to the L1 Requestor, and a REPLICATION_START message, containing both the block and the directory information, is sent to the L2 receiver. When the Receiver gets the message, allocates a cache line for the block, and replies with a REPLICATION_ACK to the L2 Sender, which will conclude the migration process with a REPLICATION_END message. If a new request is received by the L2 Sender while waiting for the REPLICATION_ACK, the request is forwarded to the L2 Destinator, which will serve it.

B. Replication protocol for shared blocks

The Re-NUCA architecture implements the replication mechanism via a message exchange between the bank that receives a request that hits the block (sender), and the bank that will have to store the replica (receiver). We assume that banks are able to distinguish between ***near*** *requests* (coming from L1s placed at the closest NUCA side) and ***far*** *requests* (coming from L1s placed at the farthest NUCA side).

The sender bank triggers the replication process under the following conditions:

1. *the request that hits the block is a* ***GETS***. In case of GETX, specific actions has to be taken in order to manage coherence, as discussed in Section III.B;
2. *the request is recognized as* ***far***. In this way, we replicate only in case of potential conflict hit;
3. *the bank is* ***NOT*** *the Collector bank* (in this case, a simple migration process can be triggered). We do this in order to avoid race conditions that arise due to the interleaving between the Collector mechanism (managing of off-chip misses), the FMA protocol, and the replication mechanism.

Figure 3 shows a sequence diagram of the actions taken by the Sender (L2-48) and the Receiver (L2-24) banks involved in a replication process. The choice of which L2 bank will host the replica is fixed for each bankset, and is always the bank next to the Collector at the opposite side of the cache. For example, referring to the bankset 0 in Figure 1, banks 0,8,16,24 will send the REPLICATION_START to bank 40, while banks 40,48,56 will send the message to bank 24. Figure 3 also highlights how subsequent requests received during the replication process are managed: when the Sender receives a new request while waiting for the REPLICATION_ACK, the request is forwarded to the

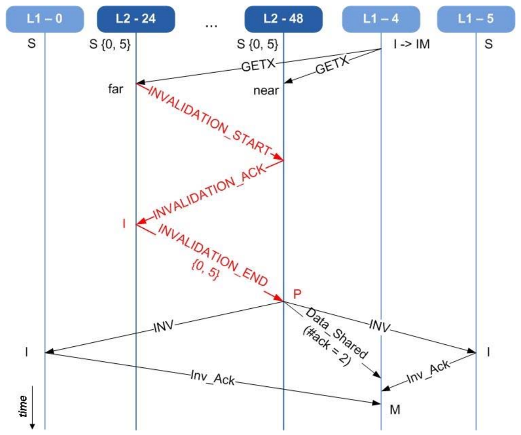

Figure 4. Sequence diagram representing the Invalidation protocol for a replica. When a GETX message is issued from L2-5, it is received by all the banks of the bankset. As the L1 requestor is L1-4, the bank L2-24 receives it as *far*, and the bank L2-48 as *near*. In this way, L2-24 knows that its copy must be invalidated, and issues an INVALIDATION_START message toward the L2-48. On the other hand, as L2-48 has received the GETX *near*, knows that an invalidation message is going to be received. When the INALIDATION_START arrives at L2-48, it replies with INALIDATION_ACK to L2-24, that issues an INALIDATION_END (containing the directory information collected so far) before invalidating his copy. When L2-48 receives the last message, it takes the proper actions according to the coherence protocol (in this case, sends the INValidation messages to all the old sharers, and changes the status to P), resets the *isReplica* bit, then provides the requestor with the most up-to-date block version.

Receiver, that will serve it. The Receiver is able to detect duplicates (similar to the FMA protocol: if the original GETS arrives at the Receiver after the REPLICATION_START, the Fwd_GETS is recognized as a duplicate), so eventually it would discard the message forwarded by the Sender.

At the end of the replication protocol, two copies of the same block exist. In order to let a copy be aware it is a replica, an extra-bit (called *isReplica*) must be added to the TAG field of the block. The *isReplica* bit is set whenever a replication process ends, and reset when a copy is no longer a replica (see Section III-C). Both the copies are able to migrate toward the faster way of the respective target side: in the example discussed above, the copy in L2-24 migrates toward L2-0, while the replica in L2-48 migrates toward L2-56. The migration process is started for a replica whenever a GETS *near* is received; GETS *far* are treated as miss.

C. *Invalidation protocol for replica blocks*

As Re-NUCA allows the replication of both Read-Only and Read-Write blocks, it is possible to receive a GETX message when the referred block is replicated. In this case, in order to guarantee the correctness of memory operations, Re-NUCA adopts an Invalidation protocol that involves both the banks that hold a copy of the block. In particular, Re-NUCA invalidates the farthest replica with respect to the L1 requestor. For example, if replicas are stored in L2-24 and L2-48 and the GETX is issued by the L1-5, then the copy in L2-24 will be invalidated, while the one in L2-48 will

Table 1. Configuration parameters

CPUs	*8 cpus (Ultra Sparc II), in-order*
Clock Frequency	*~5 GHz*
L1 cache	*Private 16 Kbytes I + 16 Kbytes D, 2 way s.a., 1 cycles to TAG, 2 cycles to TAG+Data*
L2 cache	*16 Mbytes, 64 banks (256 Kbyte banks, 4 way s.a., 5 cycles to TAG, 8 cycles to TAG+Data)*
Block Size	*64 bytes*
NoC configuration	*Partial 2D Mesh Network; NoC switch latency: 1 cycle; NoC link latency: 1 cycle; flit size: 32 bits*
Main Memory	*2 GByte, 300 cycles latency*

change its state (according to the coherence protocol) and resets the *isReplica* bit in the TAG field.

Figure 4 shows an example of the Invalidation protocol in Re-NUCA. The need of adopting such Invalidation protocol with a three messages handshake comes from the fact that, as in the case of replication, subsequent requests for the same block could arrive during the invalidation process. The management of such race conditions is exactly the same as in the previous case: requests are forwarded from the invalidating bank to the other one, that will serve them (or discard them in case they are recognized as a duplicate) according to the coherence protocol.

Note that, as the two replicas are independent, and they can migrate, none of them knows which of the banks is currently holding the other copy. For this reason, the INVALIDATION_START is sent to all the banks that can store the other copy: only the correct bank will accept the message, in all the other cases it will be discarded.

Note also that it is possible that two separate (and opposite) cores issue at the same time a GETX for their nearest replica of a given block. As Re-NUCA must guarantee that one of the replicas (the farthest one) must be invalidated, in this case both replicas would be candidates for invalidation, if they both receive the respective *near* GETX before the *far* one, thus behaving as L2-48 in Figure 4. Re-NUCA is able to manage such race condition by means of a per-bankset arbiter. The arbiter implements a two states machine per block: when the involved L2 banks receive the *far* GETX, they both send a CONTENTION message to the arbiter of the pertaining bankset, that assigns the exclusivity of the copy to the sender of the first received CONTENTION message. Hence, the L2 that looses the contention phase sends the INVALIDATION_END to the contention winner, then invalidates its copy. Once the winner is sure it is the only copy, it serves the two GETX according to the coherence protocol.

D. *Re-NUCA implementation overhead*

In order to manage replication, Re-NUCA introduces the *isReplica* bit in the TAG field of each L2 blocks. As we assume a 16 Mbytes L2 cache, with 64 bytes per block, we have 256K blocks, and thus we have 256 Kbits of extra storage overhead, that is less than 2x10^{-3} % of the overall L2 storage capacity.

The arbiter is implemented in the Collector's controller by adding extra registers, introducing an overhead of 4 bits (1 bit for representing the two states, and 3 bits for storing the ID of the winner L2 bank in the bankset) per each

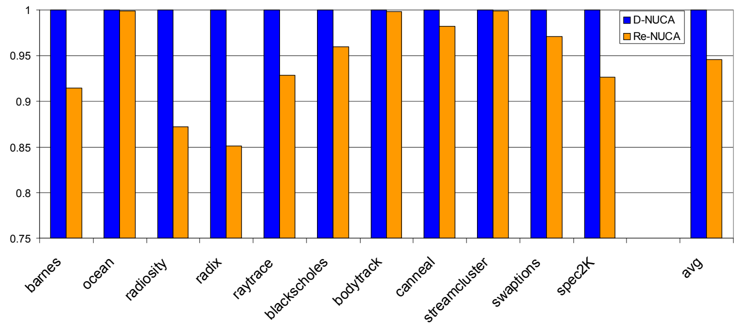

Figure 5. Normalized CPI. All the values are normalized with respect to D-NUCA

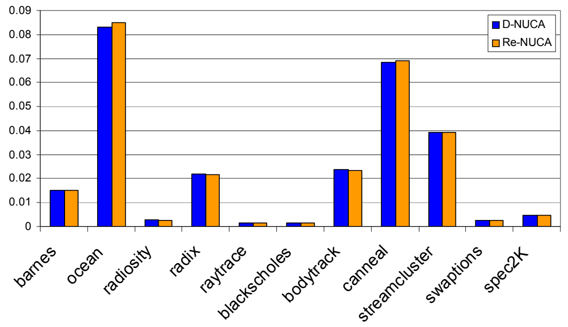

Figure 6. L2 Miss Rate.

possible contention. As we have eight processors, each arbiter of each bankset must be able to manage at most 4 contentions (i.e., it needs of 4 registers): thus the chip storage overhead introduced by the arbiters is

(8 banksets)(4 contentions)*(4 bits) = 128 bits = 16 bytes*

that is about the 10^{-6} % of the overall L2 storage capacity.

IV. METHODOLOGY

We performed full-system simulation using Simics [24]. We simulated an 8-cpu UltraSparc II CMP system, each CPU using in-order issue, running at 5 GHz. We used GEMS [25] in order to simulate the cache hierarchy: private L1s are composed by 2 ways set associate Instructions and Data caches (16 KB each), while the shared S-NUCA L2 cache is composed by 64 banks (each of 256 KB, 4 ways set associative), for a total storage capacity of 16 MB. We assumed 2 GB memory with a 300-cycle latency. Cache latencies to access TAG and TAG+Data have been obtained by CACTI 5.1 [26] for the specified nanotechnology. The NoC is organized as a partial 2D mesh network, with 64 wormhole [15,16] switches (one for each NUCA bank); NoC link latency has been calculated using the Berkeley Predictive Model [27]. Table 1 summarizes the configuration parameters for the considered CMP.

Our simulated system runs the Sun Solaris 10 operating system. We run applications from the SPLASH-2 benchmark suite [28] and from the PARSEC 2.0 suite [18]; we also run a set of applications from the SPEC2K suite [30]. All the applications were compiled with the *gcc* provided with the Sun Studio 10 suite.

V. RESULTS

We simulated the execution of *ocean*, *barnes*, *radix*, *radiosity* and *raytrace* from the SPLASH-2 suite, and *blackscholes*, *bodytrack*, *canneal*, *streamcluster* and *swaptions* from the PARSEC 2.0 suite. 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). We choose the *Cycles-per-Instruction* (CPI) as the reference performance indicator, and the *bytes-per-instruction* as the reference NoC traffic indicator.

A. *Performance evaluation*

Figure 5 shows the Normalized CPI for the considered benchmarks. We notice that in none of the considered cases, we have a performance degradation. Instead, in the most part of the applications, we have a reduction in the execution time that varies from about 2% (canneal) up to 15% (radix). On average, we have about 6% of performance improvement. This has been obtained without affecting the total L2 miss rate: as we can see in figure 6, the miss rate does not present significant variations for the considered applications. This is due to the small number of replications that are triggered by the Re-NUCA architecture with respect to the total number of requests issued by the eight L1 caches. As a consequence, we observe a good performance improvement.

To investigate the behaviors of our mechanism, we show in figure 7 the distribution of the hits in each way of the shared cache. As we can see, in the case of D-NUCA many applications (such as barnes, radix and canneal) present a hits distribution that involves no faster ways in the most part of the cases. For example, radix only hits in the faster ways (1 and 8) for 26 % of the total L2 hits. Instead, if we consider the Re-NUCA hits distribution, for the most part of the applications, we have a complete unbalance of the accesses in the faster ways. This is due to both the replication and migration mechanisms: once each shared block has been replicated, the migration mechanism, according to the frequency of the accesses to such replicas, lets them migrate towards the faster way of the pertaining target side. In this way, our mechanism is able to reduce the average L1 miss latency by reducing the hit time in L2 cache. Figure 8 shows the average L2 hit latency in both D-NUCA and Re-NUCA. With the adoption of the replication mechanism, we improve the L2 response time of about 12% on average, but for those applications that do not present a good hit distribution, we reduce the hit time of more than 25% (about 30% for radix). This is due to the fact that, as the most part of the hits are concentrated in the faster ways, even if request messages are propagated to all the bankset's banks (broadcast search as in any D-NUCA scheme), the requests blocks are provided in a very short time.

We also observe that ocean, bodytrack and streamcluster don't take advantage from the adoption of the replication

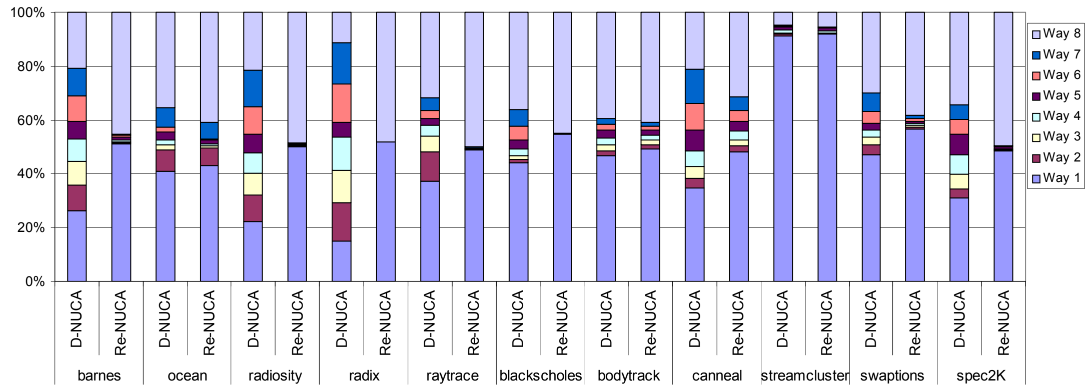

Figure 7. Hits distribution. Label of each series (Way 1, … , Way 8) is chosen according to the reference architecture of Figure 1

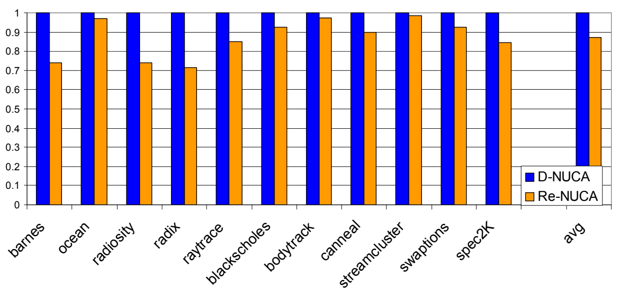

Figure 8. Normalized Average L2 Hit Latency.

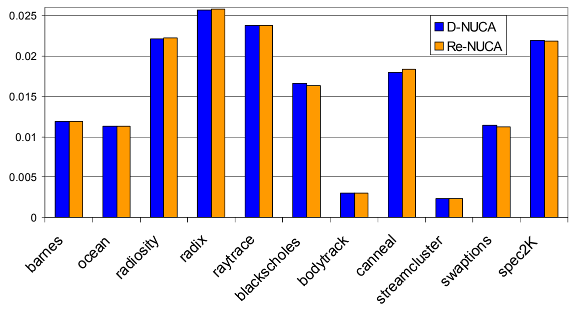

Figure 9. L1 miss rate.

mechanism. This result was expected as none of them varies the hits distribution when moving from D-NUCA to Re-NUCA. In fact, ocean presents a small number of shared blocks with respect to applications like barnes or radix: this low sharing among ocean's threads let the classical migration mechanism to be effective in bringing the most frequently used blocks in the faster ways, and the conflict hit phenomenon has no significant effects. As for bodytrack and streamcluster, we observe a very low L1 miss rate in the considered execution phases, so they can't take advantage from the reduced average response time that characterized Re-NUCA caches. The values of the L1 miss rates are shown in Figure 9.

B. *NoC traffic evaluation*

Another consequence of the imbalance of accesses introduced with the Re-NUCA is the variation of the utilization of the NoC. Figure 10 shows how both *control* and *data* messages impacts on the overall NoC traffic, in terms of *bytes-per-instructions*. In our implementation, control message size is 8 bytes (header only), while data message size is 72 bytes (8 bytes for the header, 64 bytes for the data, i.e. the memory block). The main aspect we can see is that the data component of the total NoC traffic is reduced for almost all the applications (except ocean, bobytrack and streamclusters for the same reasons we discussed above). This reduction is a consequence of the replication mechanism, and of the migration mechanism for the replica blocks. Consider that the faster ways are the bank lines that are closest to the processors. As the most part of the hits are in faster ways, the network distance (i.e. the number of network hops to be traversed by a packet to reach the receiver node when it leaves the sender node) is drastically reduced: in particular, only the horizontal links must be traversed. Consequently, as in Re-NUCA data packets must traverse a reduced number of hops with respect to D-NUCA, the data component of the global NoC traffic is reduced.

Another interesting point to notice is that the control part of the NoC traffic doesn't change. One might expect that such component would increase due to the overhead introduced by the replication and invalidation mechanisms. Actually, as we said before, the replication mechanism (and consequently the potential invalidations) is triggered a few times with respect to the total L2 accesses: once a block has been replicated, it migrates toward its target faster way, and is no longer replicated (until it is invalidated and then replicated again, in case of Read-Write blocks). As a result, the overhead of replication and invalidation handshakes has a negligible impact on the overall NoC traffic. On the other hand, we also observe that the control component is dominating the NoC traffic: this can be explained by considering that the search mechanism (broadcast request to all the banks of a bankset, a *miss* response message sent back to the L1 requestor by each bank [29]) is still the most significant part of the total NoC traffic. Reducing NoC traffic

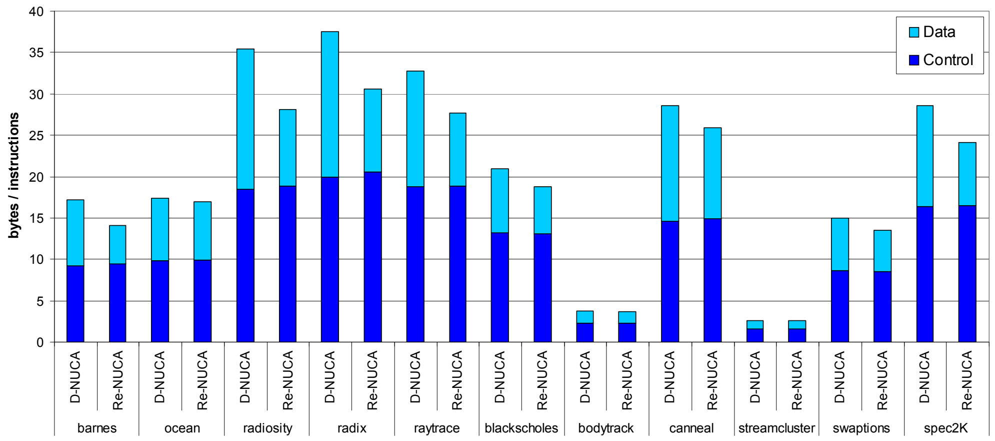

Figure 10. Total NoC traffic. Control messages are composed by 8 bytes (header only). Data messages are composed by 72 bytes (8 bytes for the header, 64 bytes for the data)

is important because dynamic energy consumption is tied to the overall network activity.

C. *Evaluation concluding remarks*

Our evaluation shows that, with respect to D-NUCA Re-NUCA improves performance and reduces NoC traffic by augmenting the number of hits in the faster ways. In particular, for those applications who strongly suffers from the conflict hits problem, we obtain significant improvement in both the CPI and average L2 hit latency, and consequently a reduction of the data component of the total NoC traffic, while introducing a negligible message overhead.

VI. Conclusion and Future Works

In this paper, we introduced Re-NUCA, a novel scheme for CMP shared caches that allows blocks replication and migration in order to reduce average cache response time and improve performance. The proposed approach ensures scalability since *i*) the *limited replication* mechanism allows only two copies of each shared block to exist in L2 cache, and *ii*) the overall NoC traffic is reduced. Results show that we obtain significant performance improvement for those applications that are affected by the conflict hit phenomenon, and an average performance gain of about 6%. The NoC traffic is decreased by reducing the network distance to be traversed by data messages. We also show that the control traffic is still dominating the overall NoC traffic: future works can focus on such aspect, as the incremental search could be adopted without introducing a significant increment in the L1 miss resolution time because the most part of the hits in a Re-NUCA cache are in the faster ways. To further improve scalability, we plain to design our system in a hierarchical fashion.

Acknowledgment

This work has been partially supported by the SARC project funded by the European Union under the contract no. 27648, by PRIN 2008 project "Design methodologies for high performance systems for scientific and multimedia applications" (contract no 200855LRP2_001), and by the FIRB project "PHOTONICA - Photonic Interconnect Technology for Chip Multiprocessing Architectures" (contract no RBFR08LE6V), both funded by the Italian MIUR.

The authors want to thank to Alberto Masia for his contribution to this work.

References

[1] K. Olukotun, B.A. Nayfeh, L. Hammond, K. Wilson and K. Chang, "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] L. Hammond, B.A. Nayfeh and K. Olukotun, "A single-chip Multiprocessor" *IEEE Computer*, 30(9), pp. 79-85, Sept. 1997

[3] K. Krewell, "UltraSparc IV Mirrors Predecessors", *Microprocessor Report*, pp. 1-3, Nov. 1997.

[4] C. McNairy and R. Bhatia, "Montecito: A Dual-Core Dual-Thread Itanium Processor" *IEEE Micro*, 25(2), pp. 10-20, Mar./Apr. 1997.

[5] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer and J. Joyner, "Power5 System Architecture" *IBM Journal of Research and Development*, 49(4), pp. 505-522, 2005

[6] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded Sparc processor" *IEEE Micro*, 25(2), pp. 21-29, Mar. 2005.

[7] 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.

[8] A. Mendelson, J. Mandelblat, S. Gochman, A. Shemer, R. Chabukswar, E. Niemeyer and A. Kumar, "CMP implementation in systems based on the Intel Core Duo processor" *Intel Technology Journal*, 10(2), pp. 99-107, 2006

[9] Z. Chishti, M.D. Powell and T.N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs" *Proc. of the 32nd annual Int. Symp. on Computer Architecture*, Madison, WI, USA, pp. 357-368, June 2005.

[10] J. Chang and G.S. Sohi, "Cooperative Caching for Chip Multiprocessors" *Proc. of the 33rd annual Int. Symp. on Computer Architecture*, Boston, MA, USA, pp. 264-276, June 2006.

[11] 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.

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

[13] 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.

[14] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, S. W. Keckler, "A NUCA substrate for flexible CMP cache sharing" *Proc. of the 19th annual Int. Conf. on Supercomputing*, Cambridge, MA, USA, pp. 31-40, June 2005.

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

[16] W.J. Dally and B. Towels, *Principles and Practices of Interconnection Networks*. San Francisco, CA. Morgan Kauffmann, Elsevier, 2004.

[17] 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

[18] C. Bienia, S. Kumar, J. Pal Singh and K. Li, "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.

[19] 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.

[20] P. Foglia, F. Panicucci, C.A. Prete and M. Solinas, "Investigating design tradeoffs in S-NUCA based CMP systems" *Proc. of the 5th annual workshop on Unique Chips and Systems*, Boston, MA, USA, pp. 53-60, 2009.

[21] P. Foglia, F. Panicucci, C.A. Prete and M. Solinas, "An evaluation of behaviors of S-NUCA CMPs running scientific workload" *Proc. of the 12th Euromicro conf. on Digital System Design*, Patras, Greece, Aug. 2009.

[22] Q. Lu, U. Bondhugula, S. Krishnamoorthy, P. Sadayappan, J. Ramanujam, Y. Chen, H. Lin and T. Ngai, "A compile-time data locality optimization framework for NUCA chip multiprocessors". *Technical Report, OSU-CISRC-6/08-TR29*

[23] 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 Europe 09*, Nice, France, pp. 598-601, Apr. 2009.

[24] Virtutec Simics, http://www.virtutech.com

[25] Winsconsin Multifacet GEMS Simulator, http://www.cs.wisc.edu/gems/

[26] S. Thoziyoor, N. Muralimanohar, J.H. Ahn and N.P. Jouppi, "CACTI 5.1" *Technical Report*, HP Laboratories, Palo Alto, CA, April 2008.

[27] Predictive Technology Model (PTM), http://www.eas.asu.edu/~ptm/

[28] 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

[29] P. Foglia, F. Panicucci, C.A. Prete and M. Solinas, "Analysis of performance dependencies in NUCA-based CMP systems" *Proc. of the 21st International Symposium on Computer Architecture and High Performance Computing*, Sao Paulo, Brazil, pp.49-56, October 2009.

[30] John L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium" *IEEE Computer*, 33(7), pp. 28-35, July 2000.

[31] J. Merino, V. Puente, J.A. Gregorio, "ESP-NUCA: A Low-cost Adaptive Non-Uniform Cache Architecture". *Proc. of the 16th IEEE Int. Symp. on High-Performance Computer Architecture*, Bangalore, India, January 2010, pp. 1-10.

[32] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki, "Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches". *Proc. of the 36th Int. Symp. on Computer Architecture*, Austin, TX, USA, June 2009, pp. 184-195

[33] M.M.K. Martin, M.D. Hill and D.A. Wood, "Token coherence: decoupling performance and correctness" *ACM SIGARCH Computer Architecture News*, 31(2), pp. 182-193, May 2003