Investigating design tradeoffs in S-NUCA based CMP systems

P. Foglia, C.A. Prete, M. Solinas
University of Pisa
Dept. of Information Engineering
via Diotisalvi, 2 56100 Pisa, Italy
{foglia, prete, marco.solinas}@iet.unipi.it

F. Panicucci
IMT Lucca
Institute for Advanced Studies
piazza San Ponziano, 6 55100 Lucca, Italy
f.panicucci@imtlucca.it

***Abstract*—A solution adopted in the past to design high performance multiprocessors systems that were scalable with respect to the number of cpus was the design of Distributed Shared Memory (DSM) multiprocessor with coherent cache, whose coherence was held by a directory-based coherence protocol. Such solution permits to have high level of performance also with high numbers of processors (512 or more). Modern systems are able to put two or more processors on the same die (Chip Multiprocessors, CMP), each with its private caches, while the last level caches can be either private or shared. As these systems are affected by the wire delay problem, NUCA caches have been proposed to hide the effects of such delay in order to increase performance. A CMP system that adopt a NUCA as its shared last level cache has to be able to maintain coherence among the lowest, private levels of the cache hierarchy. As future generation systems are expected to have more then 500 cores per chip, a way to guarantee a high level of scalability is adopting a directory coherence protocol, similar to the ones that characterized DSM systems. Previous works focusing on NUCA-based CMP systems adopt a fixed topology (i.e. physical position of cores and NUCA banks, and the communication infrastructure) for their system and the coherence protocol is either MESI or MOESI, without motivating the reasons of such choices. In this paper, we present an evaluation of an 8-cpu CMP system with two levels of cache, in which the L1s are private of each core, while the L2 is a Static-NUCA shared among all cores. We considered three different system topologies (the first with the eight cpus connected to the NUCA at the same side, the second with half of the cpus on one side and the others at the opposite side, the third with two cpus on each side), and for all the topologies we considered MESI and MOESI. Our preliminary results show that processor topology has more effect on performance and NOC bandwidth occupancy than the coherence protocol.**

I. Introduction

In the past, Distributed Shared Memory (DSM) systems with coherent caches were proposed as an high-scalable architectural solution, as they were characterized by powerful processing nodes, each with a portion of the shared memory, connected through a scalable interconnection network [1], [2]. In order to maintain high level of scalability with respect to the number of cores, the coherence protocol usually adopted in such system was a directory coherence protocol, where directory information was held at each node. Directory coherence protocols rely on message exchange between the nodes that need a copy of a given cache block, and the home node (i.e. the node in the system that has to manage directory information for the block). With the increasing number of transistors available on-chip due to technology scaling [3], multiprocessor systems have shifted from multi-chip systems to single-chip systems (Chip Multiprocessors, CMP) [4], [5], in which two or more processors exist on the same die. Each processor of a CMP system has its own private caches, and the last level cache (LLC) can be either private [6], [7] or shared among all cores [8], [9], [10], [11]; hybrid designs have been also proposed [12], [13], [14]. CMPs are characterized by low communication latencies with respect to classical many-core and DSM systems, as the signal propagation delay in on-chip lines is lower than in off-chip wires [15]. However, as clock frequencies increase as well as the delay in communication lines, signals need more clock cycles to be propagated on the chip, thus resulting in higher wire delay, and this delay significantly affects performance [5], [2]. In order to face the wire delay problem, Non-Uniform Cache Access (NUCA) architecture [16], [17], [10] has been proposed: a NUCA is a bank-partitioned cache in which the banks are connected by means of a communication infrastructure (typically, a Network-on-chip, NoC [18], [19]), and it is characterized by a non-uniform access time. NUCAs have been proved to be effective in hiding the effects of wire delay. When adopted in CMP systems, a NUCA typically represents the LLC shared among all the cores [10], [17], and all the private, lower cache levels have to be kept coherent by means of a coherence protocol; the cores in the system are able to communicate both among themselves and with NUCA banks. As NoCs are characterized by a message-passing communication paradigm, the communication among all kind of nodes in the system (i.e. shared cache banks and processor with private caches) is based on the exchange of many types of messages. In this context, the coherence protocol is implemented as a directory-based protocol, similar to those designed for DSM systems, in order to meet the same high degree of scalability. By exploiting the fact that the LLC is shared among all cores, our proposal is to adopt a non-blocking directory [20], that is distributed in NUCA banks: NUCA banks can be adopted as home nodes for cache blocks, and the directory information is stored in the TAG field of each block present in the NUCA. Previous

works proposed various CMP architectures based on NUCA cache, each adopting as the base coherence protocol either MESI [12], [17] or MOESI [10]. However, to the best of our knowledge, none of them motivated the choice of neither the coherence protocol nor the system topology; instead, we believe that the behavior of a NUCA-based CMP is heavily influenced by both these aspects. In this paper, we present a preliminary evaluation of design tradeoffs for 8-cpus CMP systems that adopt a Static-NUCA (S-NUCA) as the shared L2 cache, while each processor has its own private L1 I/D caches; the cache hierarchy is inclusive. We consider three different configurations for the CMP: the first configuration has all the processors connected to the NUCA NoC at the same side (8p), the second one presents four cpus on one side and the others on the opposite side (4+4p) and in the third one we have two cpus on each side (2+2+2+2p); the three configurations are shown in Figure 1. For all 8p, 4+4p and 2+2+2+2p we evaluate the behaviors of the system when the coherence protocol is either MESI or MOESI.

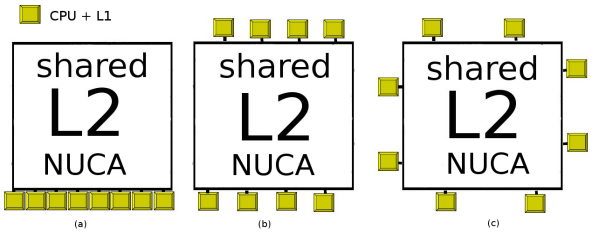

Fig. 1: *The considered CMP topologies: 8p (a), 4+4p (b) and 2+2+2+2p (c)*

The rest of this paper is organized as follows. Section 2 presents some related works. Section 3 explains the design of our MESI and MOESI protocols, and also contains some considerations about the protocols behaviors versus the system topology. Section 4 presents our evaluation methodology. Section 5 discusses our preliminary results. Section 6 concludes the paper and presents our future work.

II. Related Works

Lot of work has been proposed on DSM systems with cc-NUMA, from both the research and industrial point of view, leading to the design of computer machines that are commercial successful [1], [20], [21]. In particular, [1] and [2] propose two DSM systems in which communicating nodes contain both processor(s) and a portion of the total shared Main Memory; these nodes are connected through a scalable interconnection network. The cache coherence protocol is a directory-based MESI, removing the broadcast bottleneck that prevents scalability of broadcast. In [1] the basic node is composed by two processors, up to 4GB of main memory and its corresponding directory memory, and has a connection to a portion of the I/O system. Within each node, the two processors are not connected through a snoopy bus, instead, they operate as two separated processors multiplexed over the single physical bus. The [2] node, called cluster, also consists of a small number of high-performance processors, each with its private caches, and the directory controller for the local portion of the main memory that is interfaced to the scalable network; the cache coherence protocol within a cluster relays on a bus-based snoopy scheme, while the inter-cluster coherence protocol is directory-based. Instead, in this work we discuss a class of systems (CMPs) in which eight processors exists on the same chip, and the coherence protocol is a directory protocol in which directory information is held in the TAG field of a huge, shared on-chip cache. With respect to [1] and [2], CMP systems are characterized by low latency on-chip communication lines, making the L1-to-L1 not so expensive as in the case of DSM systems.

Beckmann and Wood in [10] propose an 8-cpus CMP system in which each processor has its private L1 I/D caches and a huge shared L2 NUCA cache; the cpu+L1 nodes are distributed all around the shared cache (two nodes for each side). The NUCA can work as both Static and Dynamic NUCA. The coherence protocol is a MOESI directory-based coherence protocol in which directory information in held in cache TAGs. The NUCA-based CMP systems we analyze in this paper vary across two different topologies, working with two different coherence protocols, MESI and MOESI.

Huh et al. in [17] propose a CMP system in which 16 processors share a 16 MB NUCA cache; half of the cpus are connected to one side of the NUCA, the other half at the opposite side. The chosen coherence protocol is a directory version of MESI, with directory information held in NUCA TAGs. They also propose an interesting evaluation of the architecture with both Static and Dynamic NUCA, and for different sharing degrees; nevertheless, the topology and the coherence protocol are fixed.

Chisthi et al. in [12] evaluate ah hybrid design that tries to take advantage from 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. The system is based on a the MESI coherence protocol, modified in order to perform Controlled Replication, and the system topology is also fixed to a 4-cpu CMP with the processors on two opposite sides of the chip.

A comparative study of coherence protocol were proposed by Martin et. al in [22], when they proposed the Token Coherence protocol. However, the Token Coherence relays on broadcast communication, so it cant reach a high degree of scalability with respect to the increasing number of processors. For this reason, we prefer to analyze conventional directory coherence protocols.

III. The MESI and MOESI Coherence Protocols

This section presents our directory-based version of both MESI and MOESI. Such kind of protocols have had a renewed relevance in the context of CMP systems, but it is difficult to find a detailed description of their characteristic in recent CMP papers. The main characteristic of our protocols is that the directory (i.e. the NUCA bank the current block is mapped to) is non-blocking (with the exception of a new request received for a L2 block that is going to be replaced). A non-blocking directory is able to serve a subsequent request for a given

block even if this is still ongoing on a previous transaction, without the need of stalling the request or nacking it [20]. Both the protocols relay on three virtual networks [18], [19], [20]: the first one (called vn0) is dedicated to requests that the L1s send to the interested L2 bank; the second one (called vn1) is used by the L2 bank to provide the requesting L1 with the needed block (L2-to-L1 transfer), but also by the L1 that has the unique copy of the block to send it to the requesting L1 (L1-to-L1 transfer); the last one (called vn2) is used by the L2 bank to forward the request received by an L1 requestor to the L1 cache that holds the unique copy of the block (L2-to-L1 request forwarding). Our protocols were designed without requiring total ordering of messages.

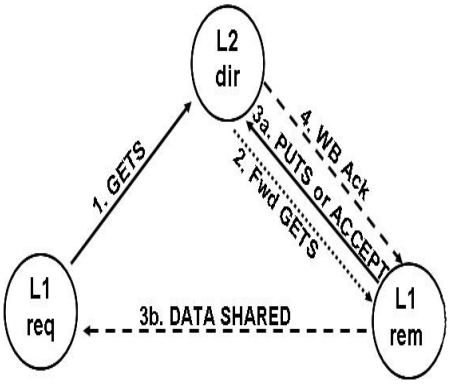

Fig. 2: *Sequence of messages in case of Load Miss, when there is one remote copy. Contiguous lines represent request messages travelling on vn0; non-contiguous lines depict response messages on vn1; dotted lines represent represent messages travelling on vn2*

In particular, vn0 e vn1 were developed without any ordering assumption, while vn2 only requires point-to-point ordering. The reason of such choice is that the performance of NUCA cache are strongly influenced by the performance (and thus by the circuital complexity) of network switches [23], [24], [25]. By utilizing wormhole flow control and static routing it is possible to design high-performance switches [19] that are particularly suited for NUCA caches.

A. MESI

The base version of our MESI protocol is similar to the one described in [20]. A block stored in L1 can be in one of the four states M (Modified: this is the unique copy among all the L1s, and the datum is dirty with respect to the L2 copy), E (Exclusive: this is the unique copy among all the L1s, and the datum is clean with respect to the L2 copy), S (Shared: this is one of the copies stored in the L1s, and the datum is clean with respect to the L2 copy) and I (Invalid: the block is not stored in the local L1). The L1 controller receives LOAD, IFETCH and STORE from the processor; in case of hit, the block is provided to the processor, and the corresponding coherence actions are taken; in case of miss, the corresponding request is build as a network message and is sent to the dedicated L2 bank through the vn0 (LOAD and IFETCH requests generate the same sequence of actions in any case, so from this moment on we consider only LOAD and STORE operations). When

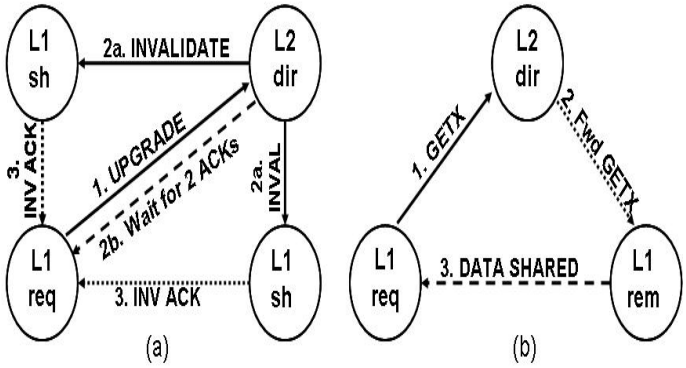

Fig. 3: *Sequence of message in case of Store Hit (a) when the block is shared by two remote L1s, and Store Miss (b) when there is one remote copy*

the L2 bank receives a request coming from any of the L1s, it can result in a hit or in a miss. In case of hit, the corresponding sequence of coherence actions is taken; in case of miss, a GET message is sent to the Memory Controller, a block is allocated to the datum, and the copy goes in a transient state [26] while waiting for the block; when the block is received from the off-chip memory, it is stored in the bank, and a copy is sent to the L1 requestor. We now discuss the actions taken by the L1 controller when a LOAD or a STORE is received, assuming that a L1-to-L2 request always hits in the L2 bank:

Load Hit. The L1 controller simply provides the processor with the referred datum, and no coherence action is taken.

Load Miss. The block has to be fetched from the L2 cache: a GETS (GET SHARED) request message is sent to the L2 home bank on the vn0, a block in the L1 is allocated to the datum, and the copy goes in a transient state while waiting for the block. When the L2 bank receives the GETS, if the copy is already shared by other L1s, the requestor is added to the sharers list, and the block is provided, marked as shared, directly by the L2 bank; if the block is present in exactly one L1, the L2 bank assumes the copy might be dirty, and the request is forwarded to the remote L1 on vn2, then the L2 copy goes in a transient state while waiting for a response. When the remote L1 receives the forwarded GETS, provides the L1 requestor with the block, then issues toward the L2 directory -on vn0- a PUTS message (PUT SHARED: it carries the latest version of the block to be sent to the bank -the L2 copy has to be updated) if the local copy was in M, or an ACCEPT message (a control message that notifies that the L2 copy is still valid) if the local copy was in E; once the L2 directory receives the response from the remote L1, updates directory information, a WriteBack Acknowledgment is sent to the remote L1, and the block is marked as Shared. Figure 2 shows this sequence of actions. Of course, if the block is not present in any of the L1s, the L2 directory directly sends an exclusive copy of the block to the L1 requestor. When the block is received by the original requestor on vn1, if it is marked as shared the copy goes in the S state, otherwise it goes in E.

Store Hit. If the block is in M, the L1 controller provides the processor with the datum, and the transaction terminates. If the block is in E, the L1 controller provides the processor with the datum, the state of the copy changes to M and the transaction terminates. If the copy is in S, the L1 controller sends a message to the L2 bank, on vn0, in order to notify it that the local copy is going to be modified, and the other shares have to be invalidated, then the copy goes in a transient state waiting for the response from the L2 bank. When the L2 bank receives the message from the L1, sends an Invalidation message to all the sharers (except the current requestor) on vn2, then clears all the sharers in the blocks directory, and sends on vn1 to the current requestor a message containing the number of Invalidation Ack to be waited for. When a remote L1 receives a Invalidation for an S block, sends on vn1 an Invalidation Ack to the L1 requestor, then the copy is invalidated. Once the L1 requestor has received all the Invalidation Acks, the controller provides the processor with the requested block, the block is modified, then the state changes to M and the transaction terminates. Figure 3 shows this situation.

Store Miss. A GETX (GET EXCLUSIVE) message is sent to the L2 bank on vn0, a cache block is allocated to the datum and the copy goes in a transient state, waiting for the response. When the L2 bank receives the GETX, if there are two or more L1 sharers for that block, the L2 bank sends the Invalidation messages to all the sharers on vn2, then sends the block, together with the number of Invalidation Acks to be received, to the current L1 requestor, on vn1; from this moment on, everything works as in the case of Store Hit of a block in the S state. If there are no sharers for that block, the L2 bank simply stores the ID of the L1 requestor in the blocks directory information and sends on vn1 the datum to the L1 requestor. If there is just one copy stored in one L1, the L2 assumes it is potentially dirty, and forwards the request on vn2 to the L1 that holds the unique copy, then updates the blocks directory information clearing the old L1 owner and setting the new owner to the current requestor. When the remote L1 receives the GETX in forwards, sends the block to the L1 requestor on vn1, then invalidates its copy. At the end, the L1 requestor receives the block, then the controller provides the processor with the datum, the state of the copy is set to M and the transaction terminates. Figure 3 depicts this sequence of actions.

L1 Replacement. In case of conflict miss, the L1 controller chooses a block to be evicted from the cache, adopting a pseudo-LRU replacement policy. If the block is in the S state, the copy is simply invalidated, without notifying the L2 bank. If the block is either in the M or in the E state, the L1 Controller sends a PUTX (PUT EXCLUSIVE, in case of M copy: this message contains the last version of the block to be stored in the L2 bank) or an EJECT (in case of E copy: this is a very small control message that simply notifies the L2 directory that the block has been evicted by the L1, but the old value is still valid). When the L2 bank receives one of those messages, updates the directory information by removing the L1 sender, updates the block value in case of PUTX, then issues a WriteBack Acknowledgment to the L1 sender; once this receives the acknowledgment, invalidates the copy.

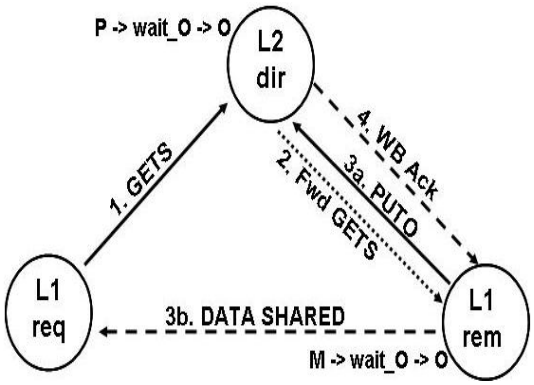

Fig. 4: *Sequence of messages in case of Load Miss, when the block is present in one remote L1, and it has been modified. The remote copy is not invalidates; instead, when the WriteBack Ack is received by the remote L1, it is marked ad Owned*

B. MOESI

The MOESI coherence protocol adopts the same four states M, E, S, and I that characterize MESI, with the same semantic meaning; the difference is that MOESI adds the state O for the L1 copies (Owned: the copy is shared, but the value of the block is dirty with respect to the copy stored in the L2 directory).

The L1 that holds its copy in the O state is called the owner of the block, while all the other sharers have their copies stored in the classical S state, and are not aware that the value of the block is dirty with respect to the copy of the L2 bank. For this reason the owner has to maintain the information of dirty copy, and update the L2 value in case of L1 Replacement. Also this MOESI coherence protocol is designed with the L2 banks that realize a non-blocking directory. Here we discuss the differences with MESI, referring to the Owner state:

Load Hit. The L1 controller simply provides the processor with the referred datum, and no other coherence action is taken.

Load Miss. The GETS message is sent to the L2 bank on vn0. When the L2 directory receives the request, if the copy is private of a remote L1 cache, the L2 bank assumes it is potentially dirty and forwards the GETS to the remote L1 through the vn2, then goes in a transient state while waiting for a response. When the remote L1 receives the forwarded GETS, if the block is dirty (i.e. in the M state) then a PUTO (PUT OWNER: this control message notifies the L2 directory that the block is dirty and is going to be owned) is sent to the L2 bank, otherwise if the block is clean (i.e. in the E state) then an ACCEPT message is sent to the L2 bank in order to notify it that the copy was not dirty, and the copy has to be considered as Shared and not Owned; in both cases, the remote L1 sends a copy of the block to the current requestor on vn1, and the L2 directory responds to the remote L1 with a WriteBack Acknowledgment, then updates the directory information by storing that the block is either owned in case of PUTO- by the remote L1 or shared in case of ACCEPT. Once

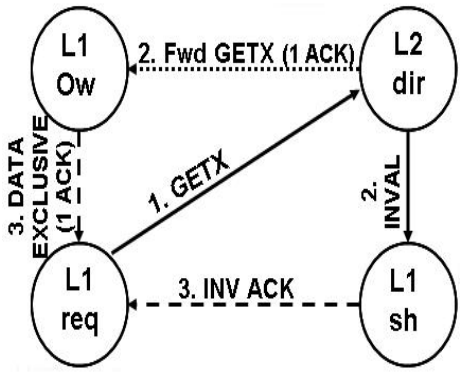

Fig. 5: *Sequence of messages in case of Store Miss when copy is Owned be a remote L1*

the L1 requestor receives the block, the controller provides the processor with the referred datum, then the copy is stored in the S state. Figure 4 illustrates this sequence of actions. If the copy was already Owned, when the L2 directory receives the GETS request, simply adds the L1 requestor to the sharers list and forwards the request to the owner, that will provide the L1 requestor with the last version of the block.

Store Hit. When a store hit occurs for an O copy, the sequence of steps is the same as in the case of a store hit for an S copy in MESI.

Store Miss. The GETX message is sent to the L2 directory through the vn0. If the block is tagged as Owned by a remote L1, the GETX is forwarded through the vn1 to the current owner (together with the number of Invalidation Acknowledgment to be waited by the L1 requestor) and an Invalidation is sent to the other sharers in the list, then the sharers list is empty and the L1 requestor is set as the new owner of the block. When the current owner receives the forwarded GETX, sends the block to the L1 requestor together with the number of Invalidation Acknowledgment that it has to wait, then the local copy is invalidated. Once the L1 requestor has received the block and all the Invalidation Acknowledgment, the cache controller provides the processor with the referred datum, then the block is modified and stored in the local cache in the M state. Figure 5 shows this case. L1 Replacement. When the L1 controller wants to replace a copy in O, sends a PUTX message to the L2 directory. Once this message has been received, the L2 bank updates the directory information by clearing the current owner, then stores the new value of the block in its cache line, and sends a WriteBack Acknowledgment to the old owner (from this moment on, the block is supposed to be Shared as in the case of MESI). When the owner receives the WriteBack Acknowledgment, invalidates its local copy.

IV. Methodology

We performed full-system simulation using Simics [27]. We simulated an 8-cpu UltraSparc II CMP system, each cpu using in-order issue, running at 5 GHz. We used GEMS [28] in order to simulate the cache hierarchy and coherence protocols: private L1s have 64 KB of storage capacity, 2 ways set associate Instructions and Data caches (32 KB each), while the shared S-NUCA L2 cache is composed by 256 banks (each of 64 KB, 4 ways set associative), for a total storage capacity of 16 MB; we assumed Simple Mapping, with the low-order bits of index determining the bank [16]. We assumed 2 GB of main memory with a 300-cycle latency. Cache latencies to access TAG and TAG+Data have been obtained by CACTI 5.1 [29] for the specified nanotechnology (65 nm). The NoC is organized as a partial 2D mesh network, with 256 wormhole [18], [19] switches (one for each NUCA bank); NoC link latency has been calculated using the Berkeley Predictive Model.

Number of CPUs	8
CPU type	UltraSparcII
Clock Frequency	5 GHz (16 FO4 @ 65 nm)
L2NUCA Cache	16 MB, 256 x 64KB, 16 ways s.a.
L1 cache	Private 32 Kbytes I + 32 Kbytes D, 2 way s.a., 3 cycles to TAG, 5 cycles to TAG+Data
L2 cache	16 Mbytes, 256 banks (64 Kbyte banks, 4 way s.a., 4 cycles to TAG, 6 cycles to TAG+Data
NoC configuration	Partial 2D Mesh Network; NoC switch latency: 1 cycle; NoC link latency: 1 cycle
Main Memory	2 GByte, 300 cycles latency

TABLE I: *Simulation Parameters*

Table I 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 [30] benchmark suite, compiled with the gcc provided with the Sun Studio 10 suite. Our simulations run until run completion, with a warm-up phase of 50 Million instructions.

V. Results

We simulated the execution of different benchmarks from the SPLASH-2 suite running on the three different systems (8p, 4+4p and 2+2+2+2p) we introduced before. We chose the Cycles-per-Instruction (CPI) as the reference performance indicator.

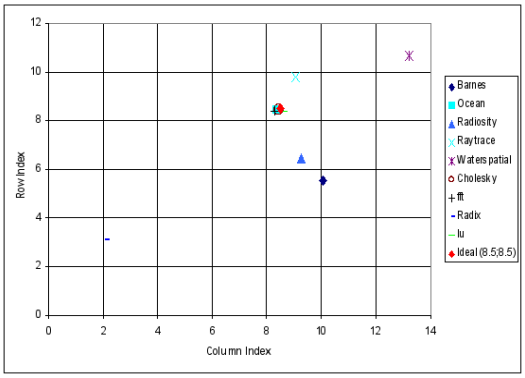

Fig. 6: *Coordinates of the accesses baricentres*

Figure 7 shows the normalized CPI for the considered configurations. As we can see, the performance difference between MESI and MOESI is very little (less than 1%) in all cases (except cholesky that presents a performance degradation of about 2% in the 4+4p configuration). This is a consequence of the very little number of L1-to-L1 block transfers with

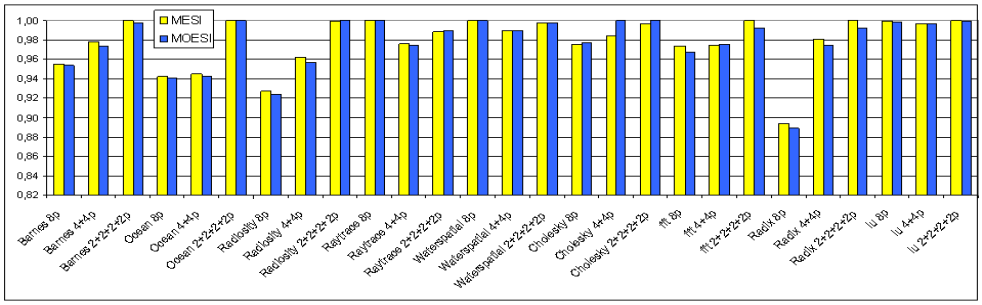

Fig. 7: *Normalized CPI*

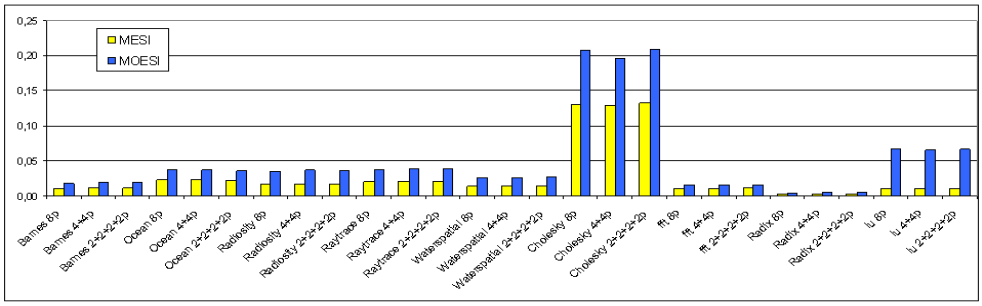

Fig. 8: *(L1-to-L1 block transfers / L1-to-L2 total requests) Ratio*

respect the total L2 accesses, as Figure 8 demonstrates (about 5% on average). Cholesky has a great number of L1-to-L1 transfers (15% for MESI, 20% for MOESI) that lead to a performance degradation in the 4+4p configuration for MOESI: the requests reach the directory bank of the S-NUCA, then 20% of them have to be forwarded to another L1 owner; if we have half of the L1s on the opposite side of the chip, the communication paths are longer, thus latencies are increased. Figure 9 shows the normalized L1 miss latency, and highlights the contribution of each type of transfer. As one might expect, the contribution of L1-to-L1 transfers is very little in all the considered cases (in cholesky the L1-to-L1 transfer has the bigger impact), and the miss latency is dominated either by L2-to-L1 block transfers or by the case of L2 miss. Figure 7 also shows that performance are in some cases affected by topology changes: to investigate this aspect, we calculated the baricentres of the accesses for each benchmark as a function of the accesses to each S-NUCA bank; the baricentres are reported in Figure 6. As the ideal case (i.e. a completely uniform accesses distribution) has the baricentre in (8.5;8.5), Figure 6 shows that there are three classes of applications, having the baricentre i) very close to the ideal case (e.g., ocean and lu), ii) in the lowest part of the S-NUCA (e.g., radix and barnes), or iii) in the highest part of the shared cache (e.g., raytrace and waterspatial). We observe three different behaviors: the ocean class of the applications dont present a significant performance variation when moving from 8p to 4+4p, but the 2+2+2+2p configuration performs worst then the others (except for lu); cholesky presents a little performance degradation also for 4+4p, even if its baricentre is very close to the ideal case: this phenomenon is due to the great impact of L1-to-L1 transfers, that have to travel along longest paths. The radix class has a greater performance degradation when moving to 4+4p and 2+2+2+2p, as the most part of the accesses are in the bottom of the shared NUCA, so moving half (or more) of the cpus to distant sides of the NUCA leads to an increase of the NUCAs response time. Finally, the raytrace class performs better with the 4+4p topology, as the most part of the accesses are in the top of the shared NUCA. Figure 10 shows the percentage of NUCA bandwidth utilized by the considered applications. When the baricentre is in the bottom of the cache, the bandwidth occupancy increases for 4+4p and 2+2+2+2p topology, but when the baricentre is in the top, the utilization decreases. For those applications similar to the ideal case, the utilization doesnt change for 8p and 4+4p, but increases for the 2+2+2+2p (except for cholesky). Having a low bandwidth utilization is impor- tant as dynamic power budget is tired to the traffic traveling on the NoC. In conclusion, topology changes have a greater influence on both performance and NoC traffic than the choice of the coherence protocol.

VI. Conclusion and Future Works

We presented a preliminary evaluation for two different coherence strategy, MESI and MOESI, in a 8-cpus CMP system with a large shared S-NUCA cache where the topology vary across three different configurations (i.e. 8p, 4+4p and 2+2+2+2p). Our experiment show that CMP topology has a great influence on performances, instead the protocol has not. Future works will present a full benchmarks evaluation of this aspect for S-NUCA based systems and also for D-NUCA based CMP architectures. The aim is to find an architecture which is mapping independent for general purpose applications and to exploit the different mappings strategy to increase performances in the case of specific applications.

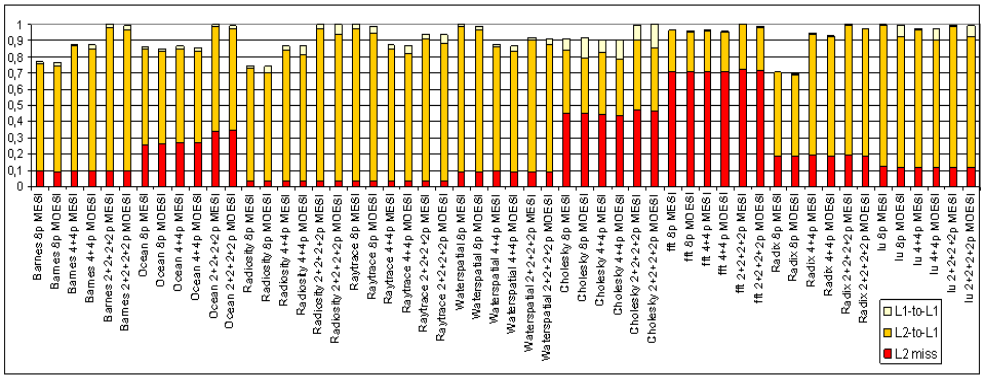

Fig. 9: *Breakdown of Average L1 Miss Latency (Normalized)*

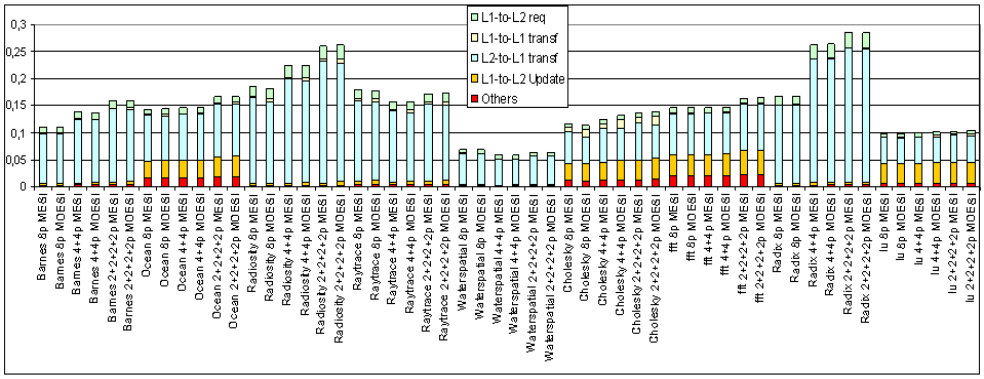

Fig. 10: *Impact of different classes of messages on total NoC Bandwidth Link Utilization (%)*

ACKNOWLEDGMENT

This paper has been supported by the HiPEAC European Network of Excellence (www.hipeac.net) and has been developed under the SARC European Project on Scalable Computer Architecture (www.sarc-ip.org).

REFERENCES

[1] J. Laudon and D. Lenoski, "The sgi origin:a ccnuma highly scalable server," *Proceedings of the 24th international symposium on Computer architecture*, pp. 241–251, 1997.

[2] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, "The directory-based cache coherence protocol for the dash multiprocessor," *Proceedings of the 17th international symposium on Computer Architecture*, p. 148, 1997.

[3] "International technology roadmap for semiconductors. semiconductor industrial association, 2005."

[4] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," pp. 2–11, 1996.

[5] L. Hammond, B. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," *IEEE Computer*, vol. 30, no. 9, 1997.

[6] K. Krewell, "Ultrasparc iv mirrors predecessors," Microprocessor Report, Nov. 1997.

[7] C. McNairy and R. Bhatia, "Montecito: A dual-core dual-thread itanium processor," *IEEE Micro*, vol. 25, no. 2, pp. 10–20, 1997.

[8] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner, "Power5 system architecture," *IBM Journal of Research and Development*, vol. 49, no. 4, 2005.

[9] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded sparc processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, 2005.

[10] B. Beckmann and D. Wood, "Managing wire delay in large chip-multiprocessor caches," *IEEE Micro*, Dec. 2004.

[11] Mendelson, Mandelblat, Gochman, Shemer, Chabukswar, Niemeyer, and Kumar, "Cmp implementation in systems based on the intel core duo processor," *Intel Technology Journal*, vol. 10, 2006.

[12] Z. Chishti, M. Powell, and T. Vijaykumar, "Optimizing replication, communication, and capacity allocation in cmps," *Proceedings of the 32nd annual international symposium on Computer Architecture*, pp. 357–368, 2005.

[13] J. Chang and G. Sohi, "Cooperative caching for chip multiprocessors," *Proceedings of the 33rd annual international symposium on Computer Architecture*, pp. 264–276, 2006.

[14] M. Zhang and K. Asanovic, "Victim replication:maximizing capacity while hiding wire delay in tiled chip multiprocessors," *Proceedings of the 32nd annual international symposium on Computer Architecture*, pp. 336–345, 2005.

[15] M. Ho and Horowitz, "The future of wires," *Proc. of the IEEE*, vol. 89, pp. 490–504, 2001.

[16] C. Kim, D. Burger, and S. W. Keckler, "Nonuniform cache architectures for wire-delay dominated on-chip caches," *IEEE Micro*, Nov./Dec. 2003.

[17] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A nuca substrate for flexible cmp cache sharing," *Proc. of the 19th annual int. conf. on Supercomputing*, pp. 31–40, 2005.

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

[19] Dally and Towels, *Principles and Practices of Interconnection Networks*. Morgan Kauffmann,Elsevier, 2004.

[20] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren, "Architecture and design of alphaserver gs320," *Proc. of the 9th int. conf. ASPLOS*, pp. 13–24, 2000.

[21] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach. 4th edition*. Morgan Kauffmann,Elsevier, 2007.

[22] M. Martin, M. Hill, and D. Wood, "Token coherence:decoupling performance and correctness," *ACM SIGARCH Computer Architecture News*, vol. 31, no. 2, pp. 182–193, 2003.

[23] A. Bardine, M. Comparetti, P. Foglia, G. Gabrielli, and C. A. Prete, "Nuca caches: Analysis of performance sensitivity to noc parameters," *Proc. of the Poster Session of the 4th Int. Summer School on Advanced Computer Architecture and Compilation for Embedded Systems*, 2008.

[24] ——, "On-chip networks: Impact on the performances of nuca caches," *Proceedings of the 11th EUROMICRO Conference on Digital System Design*, 2008.

[25] ——, "Performance sensitivity of nuca caches to on-chip network parameters," *Proceedings of the 20th International Symposium on Computer Architecture and High Performance Computing*, 2008.

[26] D. Sorin, M. Plakal, A. Condon, M. Hill, M. Martin, and D. A. Wood, "Specifying and verifying a broadcast and multicast snooping cache coherence protocol," *IEEE Transaction on Parallel and Distributed Systems*, vol. 13, no. 6, pp. 556–578, 2002.

[27] "Simics: full system simulation platform," `http://www.simics.net/`.

[28] "Winsconsin multifacet gems simulator," `http://www.cs.wisc.edu/gems/`.

[29] "Cacti 5.1: cache memory model," `http://quid.hpl.hp.com:9082/cacti/`.

[30] Woo, Ohara, Torrie, Singh, and Gupta, "The splash-2 programs: characterization and methodological considerations," *roceedings of the 22th Internationl Symposium on Computer Architecture*, pp. 24–36, 1995.