EMC^2: Extending Magny-Cours Coherence for Large-Scale Servers

Alberto Ros*, Blas Cuesta*, Ricardo Fernández-Pascual†, María E. Gómez*,
Manuel E. Acacio†, Antonio Robles*, José M. García† and José Duato*
**Department of Computer Engineering*
Universidad Politécnica de Valencia, 46021 Valencia (Spain)
Email: {aros,blacuesa,megomez,arobles,jduato}@gap.upv.es
†*Departamento de Ingeniería y Tecnología de Computadores*
Universidad de Murcia, 30100 Murcia (Spain)
Email: {rfernandez,meacacio,jmgarcia}@ditec.um.es

Abstract

The demand of larger and more powerful high-performance shared-memory servers is growing over the last few years. To meet this need, AMD has recently launched the twelve-core Magny-Cours processors. They include a directory cache (Probe Filter) that increases the scalability of the coherence protocol applied by Opterons, based on coherent HyperTransport interconnect (cHT). cHT limits up to 8 the number of nodes that can be addressed. Recent High Node Count HT specification overcomes this limitation. However, the 3-bit pointer used by the Probe Filter prevents Magny-Cours-based servers from being built beyond 8 nodes. In this paper, we propose and develop an external logic to extend the coherence domain of Magny-Cours processors beyond the 8-node limit while maintaining the advantages provided by the Probe Filter. Evaluation results for up to a 32-node system show how the performance offered by our solution scales with the increment in the number of nodes, enhancing the Probe Filter effectiveness by filtering additional messages. Particularly, we reduce runtime by 47% in a 32-die system respect to the 8-die Magny-Cours system.

1. Introduction and Motivation

Over the last years, the server market is experiencing a growing demand in the IT sector. This is due to both the dramatic increase in the number of electronic devices connected to the Internet and the introduction of new services and applications into the markets supported by the Internet servers. As a consequence, the number of transactions per time unit that must be processed by Internet servers is growing exponentially, which creates demand for increasingly larger and more powerful high-performance servers. Additionally, there is a growing trend by large companies to outsource their IT services (web servers, large enterprise databases, etc), contracting with Data Processing Centers the services they need. They usually offer their services to these companies by means of the virtualization of large servers, which require huge computational and storage capabilities.

Nowadays, most of these servers are based on clusters of PCs, since their architecture offers an excellent relation cost/performance. They usually rely on message-passing communications for remote memory accesses. However, message-passing increases not only the communication latencies, but also the difficulties to develop applications. This increasing programming complexity is unreasonable in the server field.

On the other hand, leading processor-manufacturer companies, such as AMD and Intel, have recently begun to include advanced interconnect technologies into their processors (Coherent HyperTransport [1] in AMD Opterons and QuickPath Interconnect [2] in Intel Nehalem processors). Unlike previous high-performance interconnects for clusters (e.g., InfiniBand [3]), these new network technologies are embedded into the processor, which allows them to directly access the memory controller. Moreover, including network interfaces into the processor chip itself enables a glueless point-to-point connection between the processors and memory controllers, providing low latency to remote memory accesses. In addition, these technologies usually provide support for memory coherency. In particular, AMD has recently launched the six- and twelve-core versions of its Opteron processors, codenamed Istanbul and Magny-Cours [4], respectively. Besides increasing the number of cores per processor package (actually, cores in Magny-Cours are distributed in two dies), the main difference with the previous generation of Opteron processors is the inclusion of a directory cache, called *HT Assist Probe Filter*

(HTA) [5], whose main aim is to reduce the number of messages generated by the cache coherence protocol. The Magny-Cours protocol, which is an adaptation of the protocol defined by the coherent HyperTransport (cHT) specification, allows to build a small cache-coherent shared memory multiprocessor (up to eight dies) in a single board.

The addition of the HTA reduces cache miss latency and coherence traffic, thereby increasing the scalability of the protocol. However, the HTA suffers the addressing limitations imposed by the cHT specification, which limits the coherence domain for Istanbul and Magny-Cours processors up to 8 dies (or nodes) [4]. This goes against the current commercial interest in developing cluster-based HPC systems able to offer large cache-coherent shared memory address spaces, such as the SGI Ultraviolet (Altix UV) [6] machines and the 3Leaf Systems DDC-server [7].

The addressing limitation of the cHT specification is solved in the new High Node Count (HNC) HyperTransport specification [8], which extends the cHT specification by encapsulating standard cHT messages into HNC packets. However, as current Opteron processors do not implement natively this extension, the coherence domain remains limited to 8 dies being required an external logic to overcome this limitation.

In this work, we present a device, called *bridge chip* or EMC^2 *chip*, that (1) provides a way to efficiently extend the coherence domain provided by the new generation of AMD Opteron processors beyond the 8-die limit, (2) maintains the advantages provided by the HTA, and (3) filters additional coherence traffic to enhance the HTA effectiveness.

The EMC^2 chip is added to each board in the system, replacing one of the existing dies. It manages the communication between dies in different boards by performing conversions between cHT and HNC packets. In this way, and unlike other extensions (e.g., Horus [9], which was aimed to extend the coherence domain for previous-generation AMD Opteron processors), our proposal agrees with the new HNC standard specification.

We have proposed three different implementations for the EMC^2 chip that cover a wide set of trade-offs between their area requirements and the amount of traffic filtered by them. Additionally, to improve the scalability of our design, we have proposed two approaches that (1) reduce the number of replacements in the HTA and (2) increase the number of remote messages allowed simultaneously in a particular board.

Simulation results show that our proposal allows to build large-scale shared-memory servers based on the new-generation Opteron processors, able to exploit the advantages of the HTA at the overall system level. Particularly, the bridge chip named as *EMC^2-OXSX* reduces the average execution time of the evaluated applications by 47% for a 32-die system respect to the 8-die system allowed by Magny-Cours, while obtaining an excellent compromise between area and traffic requirements.

Notice the two main advantages of extending the coherence domain. First, data center servers supporting virtualization solutions could use system resources in a more flexible and efficient way, allowing to define larger virtual domains and fitting better the application requirements. Second, it would allow to support HPC applications that currently can only be used in supercomputers and cluster-based computing platforms.

The remainder of this paper is organized as follows. Section 2 outlines the Magny-Cours cache coherence protocol. We present our proposals for extending AMD Magny-Cours cache coherence capabilities in Section 3. Section 4 discusses some scalability issues. We describe our simulation environment in Section 5 and present the evaluation results in Section 6. Finally, we draw conclusions in Section 7.

2. AMD Magny-Cours Cache Coherence Support

AMD Opteron processors use the cache coherence protocol defined by the cHT specification [1]. This protocol was designed to perform efficiently in a system with a small number of processors connected with tightly-coupled point-to-point HyperTransport links. It can be described as a hybrid between a snoopy and a directory protocol. It is similar to a snoopy protocol in the sense that all the coherence nodes see all transactions. However, like directory protocols, it does not rely on a shared bus and can, in fact, be characterized as a directory-based protocol without directory information, also known as Dir$_0$B [10]. This lack of directory information reduces the memory overhead and avoids the latency of accessing it.

Accesses to memory blocks are serialized by their home node (memory controller), which will broadcast messages known as *Broadcast Probe* (BP). Nodes reply to BPs with *Probe Response* (PR) messages, which are collected by the requester. Once the request is satisfied, the requester sends a *Source Done* (SD) to the home node, which is allowed to proceed with the next request for the block. The required BPs do not excessively increase bandwidth consumption in small systems. However, as the number of nodes grows, both the bandwidth consumed and the time required to receive and process all the PRs increase dramatically. Finally, writebacks of dirty blocks are sent to their home node which will reply to the requester with a *Target Done* (TD) message. Like in the previous case, the transaction ends with a SD.

Recent Istanbul and Magny-Cours processors include a small on-chip directory cache [5] called *HT*

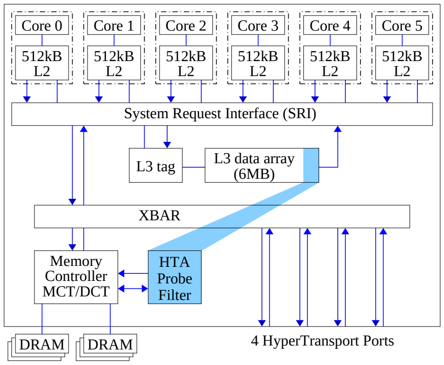


Figure 1: Block diagram of Magny-Cours dies [4].

Assist Probe Filter (HTA). Figure 1 shows the block diagram of a Magny-Cours die.

The HTA holds an entry for every block from the home node cached in the system. Each entry has 4 bytes which are used to store a *tag*, a *state* (EM, O, S1 or S)[1], and a pointer to the current *owner* of the block (3 bits). This information is used to (1) filter unnecessary BPs when no copy of the data is cached and (2) to replace some BPs with unicast *Directed Probe* (DP) messages. In case of a DP, only one response, called *Directed Response* (DR), is generated. Upon a miss on the HTA, a new entry must be allocated, which may require to replace an existing one. Before performing the replacement, all the cached copies of the block identified by the replaced entry must be invalidated either by a DP (if the replaced entry is in EM or S1 state) or by a BP (if it is in O or S state).

As Figure 1 depicts, a portion (1MB of 6MB available) of the L3 cache is dedicated to HTA entries to avoid adding a large overhead in uniprocessor systems. This provides enough space for 256K entries organized in 64K 4-way sets, which are enough for tracking 16MB (256K entries × 64 bytes/block) of data cached in the system.

Since the cHT packet format assumes 3-bit fields to identify coherent nodes, Magny-Cours systems are still limited to 8 dies. The HNC HyperTransport specification addresses this last problem by extending the cHT specification. To this end, it defines the concept of *nest* as any addressable entity (which can be anything from a single processor up to a motherboard containing several processors) and an extended packet format that can encapsulate standard cHT messages and uses a nest-based addressing scheme. However, it does not establish how packets should be handled when they move between local and remote domains. Besides, the HTA imposes an additional limitation because the pointer used to encode the current owner of a cached block has only 3 bits, bounding the Magny-Cours systems to a maximum of 8 dies. To overcome these two problems we propose the EMC^2 chip described in the next section.

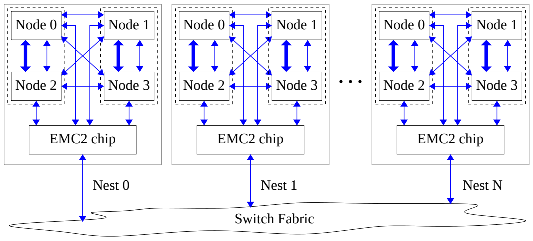


Figure 2: Overview of the proposed system. Thick arrows inside the boards represent x16 cHT links while the narrow ones are x8 cHT links.

3. Extending AMD Magny-Cours Cache Coherence Capabilities

We assume the system illustrated in Figure 2. As shown, it comprises several processor boards (referred to as nests). Each nest contains 4 processor dies (referred to as nodes) and the EMC^2 bridge chip which acts as (1) a network interface controller for the entire nest, (2) a translator between cHT and HNC packets, and (3) an extension of the HTAs of the nodes. Each board includes a consecutive fraction of the physical memory addresses. A modified BIOS or operating system can be in charge of configuring this mapping. In this paper we assume that all the boards have the same amount of memory.

3.1. Extending the Coherence Domain

To maintain coherence between nodes in different nests, we propose the use of the EMC^2 chip, whose block diagram is shown in Figure 3. From the point of view of nodes, the EMC^2 chip is seen as another node inside the nest. The EMC^2 chip and all the nodes in a nest are fully connected through a cHT interconnect. The different nests are connected by an InfiniBand switch fabric and they communicate using HNC packets encapsulated into InfiniBand packets.

Every cHT packet conveys the information of the transaction it belongs to: the node that initiated the transaction (SrcNode), its unit (SrcUnit), and a tag (SrcTag). When the EMC^2 chip has to translate a cHT packet into a HNC packet it must include the information about the associated transaction. To this end, it adds to the previous information the nest (SrcNest) where the SrcNode is located. In this way, transactions can also be unequivocally identified outside a nest.

On a HNC packet arrival belonging to an external transaction (i.e., that initiated outside the nest), the EMC^2 chip has to forward it inside the nest as a cHT packet. To avoid conflicts with the existing cHT packets belonging to internal transactions (initiated inside

[1] Blocks are stored in caches according to the MOESI states [11].

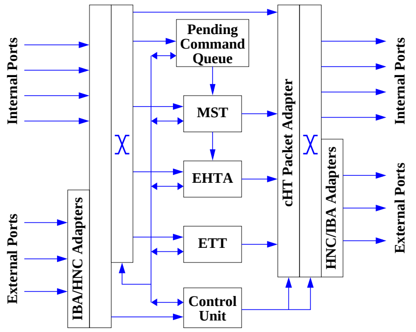


Figure 3: Block diagram of the EMC^2 chip.

the nest), the EMC^2 chip associates the generated cHT packet with a new internal transaction that will have a new SrcTag local to its nest and a new SrcNode (the EMC^2 chip identifier itself). On the other hand, when a cHT packet for that new transaction arrives to the EMC^2 chip, the HNC packet to which it is translated restores the original identifiers of the external transaction. To support these operations, the *Matching Store Table* (MST) included in the EMC^2 chip (see Figure 3) keeps the matching between the identifiers of external transactions and those of the internal ones. The number of MST entries, and consequently, the number of external transactions simultaneously in progress in the nest, is bounded by the maximum number of tags that can be generated by the cHT specification (i.e., 32 tags). When the MST is full and a new entry cannot be allocated, the incoming packets will have to be temporally stored in the *Pending Command Queue*.

The MST entries created by Broadcast/Directed Probes are valid until the associated response goes back to the EMC^2 chip. However, the entries allocated by requests remain until the arrival of the corresponding Source Done. Due to the limited number of MST entries, if every MST in the system was full of the entries allocated by requests, a deadlock scenario could occur. This is because probes would be unable to allocate new entries, and therefore, the pending requests would never complete. To avoid it, the MST must reserve at least one entry for Broadcast/Directed Probes.

Another function of the EMC^2 chip is to collect all the responses received as a consequence of a probe and transforming them into just one response packet (if needed). To accomplish this task, the EMC^2 chip uses the MST. However, given that MST entries are only allocated on the arrival of messages belonging to external transactions, we need another table that helps the EMC^2 chip to collect the responses associated to internal transactions. This additional table is called *Extended Tag Table* (ETT) and it has 512 entries (32 tags/node × 4 units/node × 4 nodes/nest). Unlike the MST, it is able to store all the transactions requesting an entry in it.

3.2. Extending the HTA Functionality

To maintain and extend the functionality of the HTAs as well as to reduce the generated coherence traffic, every EMC^2 chip includes a directory cache called *Extended HTA* (EHTA), as shown in Figure 3. An EHTA tracks the memory blocks whose home is located in its nest and that may also be cached in a remote node (outside its nest). However, the EHTA is not aware of the blocks only cached inside its nest.

Since a HTA only knows the existence of the nodes inside its nest, when the owner is in a remote node, the HTA will think that it is cached by the EMC^2 chip. Therefore, the EHTA included in the EMC^2 chip will be in charge of tracking the actual location of the owner, that is, its nest (*ownerNest* field) and node (*ownerNode* field) identifiers. Given that there are four HTAs per nest and each one holds 256K entries, we will assume 1M entries for each EHTA (64K 16-way sets). Doing so will prevent EHTA from limiting the number of blocks that can be simultaneously cached outside the home nest below the limit imposed by the local HTAs themselves.

In addition to the information of the owner, the EHTA also includes some information that helps it in the traffic filtering task. In order to cover a wide set of trade-offs between area requirements and amount of filtered traffic, we propose three different EHTA configurations.

- The EMC^2 chip with the first configuration, called *EMC^2-Base*, includes an EHTA that implements the same states as the HTA: EM, O, S1, S. These states, that require just two bits per entry, are the only information used to filter coherence traffic.
- The second configuration, assumed by the *EMC^2-OXSX* chip, adds two additional states, OX and SX, which will require three bits for codifying all the states. These new states are particularly intended to turn Broadcast Probes into Directed Probes when all the remote copies of a certain block are located in the same nest. Notice that on the arrival to the remote nest, the Directed Probe will be turned again into a Broadcast Probe in case of having to invalidate more than one copy.
- The *EMC^2-BitVector* chip, which includes the third EHTA configuration, adds a bit-vector for each EHTA entry to the first configuration. The bit-vector includes a bit for every remote nest in the system, indicating if associated block is cached or not in the corresponding nest. This allows replace Broadcast Probes with multicast probes. Although it is the configuration that filters more traffic, it needs one extra bit per remote nest for each EHTA entry what makes it the most area-demanding approach.

Since the three EHTA configurations are quite similar and in order to shorten their detailed explanation,

Table 2: Scenarios depending on the HTA state (rows) and the EHTA state (columns). *in*/*out* refers to inside/outside the home nest, and *ld*/*st* to load/store. *DP** means that the BP turns into a DP, but only while the DP is transmitted between nests. However, when the DP reaches a nest, the DP is turned into a BP (only inside that nest).

	EM	OX	O	S1	SX	S	I
EM	owner out no copy in/out ld / st:DP→DP	-	-	-	-	-	owner in no copy out ld / st: -
O	owner out no copy out copies in ld:DP→DP **st:BP→DP**	owner out copies in owner nest copies in ld:DP→DP **st:BP→DP***	owner out copies out copies in ld:DP→DP st:BP→BP	owner in 1 copy out copies in ld: - **st:BP→DP**	owner in copies out (1 nest) copies in ld: - **st:BP→DP***	owner in copies out copies in ld: - st:BP→BP	owner in no copy out copies in ld: - **st:BP→Filtered**
S1	-	-	-	owner in memory 1 copy out no copies in ld: - / st:DP→DP	-	-	owner in memory no copy out 1 copy in ld / st: -
S	-	-	-	owner in memory 1 copy out copies in ld: - / **st:BP→DP**	owner in memory copies out (1 nest) copies in ld:- / **st:BP→DP***	owner in memory copies out copies in ld: - / st:BP→BP	owner in memory no copy out copies in ld:- / **st:BP→Filtered**
I	-	-	-	-	-	-	owner in memory no copy in/out ld / st: -

Table 1: EHTA States of the *EMC*^{2}*-OXSX* chip.

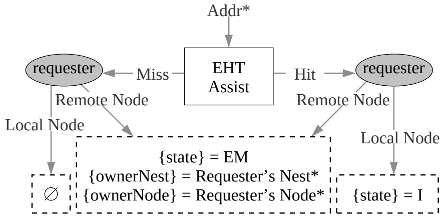
State	Description
EM	Only the owner's copy is cached outside the home nest. Other copies may be cached inside.
OX	The owner's copy is cached outside the home nest. Other copies may be cached either in the home nest or in the owner nest.
O	The owner's copy is cached outside the home nest. Other copies may be cached in any nest.
S1	At most one shared copy is cached outside the home nest.
SX	Only shared copies cached outside the home nest, all of them located in the same nest.
S	Only shared copies cached outside the home nest. They can be located in any nest.
I	No valid copy of the block cached outside the home nest.

from now on, we will just focus on the one included in the *EMC*^{2}*-OXSX* chip because it is the one that achieves a better traffic-area trade-off, as we will discuss in Section 6. In this sense, the possible block states in the EHTA are shown in Table 1.

Table 2 depicts the different scenarios that can appear depending on the block state in both the EHTA and the HTA. For each combination, it shows a short description of how the block is cached and the actions performed (if any) under load and store transactions. The three possible actions are: (1) no action, (2) turning a Broadcast Probe into a Directed Probe, and (3) filtering a Broadcast Probe. Notice that the bold actions entail a reduction in coherence traffic.

The state and the content of the EHTA is updated when the caching of the blocks changes. To do this appropriately, the EMC^2 chip uses the packets issued by the nodes located inside its nest and belonging to transactions for memory blocks mapped to that nest.

3.2.1. Broadcast Probes and Probe Responses. To update the EHTA while avoiding races, the EMC^2 chip uses the last received packet among the Broadcast Probes and Probe Responses generated as a result of a store transaction. Upon its receipt, the EMC^2 chip carries out the actions shown in Figure 4. As depicted, if there is no valid entry for that block in the EHTA (EHTA miss) and a copy is going to be sent outside the home nest (the requester is a remote node), a new entry is allocated, setting the state to EM and the block's owner (ownerNest and ownerNode) to the requester node. If the EHTA already contains an entry for the block, the EMC^2 chip updates the existing entry accordingly. Finally, when the requester is in the home nest, the EHTA entry is set to invalid because all external copies are invalidated.



* Retrieved from either the Broadcast Probe or the MST

Figure 4: Updating the EHTA by BPs or PRs for store transactions.

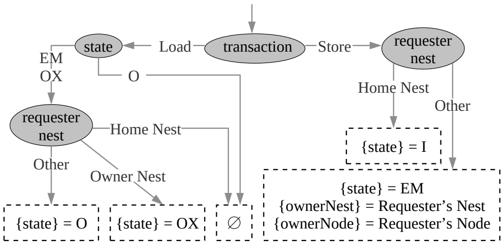


Figure 5: Updating the EHTA by DPs.

3.2.2. Directed Probes. Figure 5 shows how the EHTA is updated on a Directed Probe (DP) arrival. If an EMC^2 chip receives a DP (from inside the nest) due to a load transaction, then it means that the block's owner is outside the home nest and, therefore, the EHTA state can only be EM, OX, or O. In this

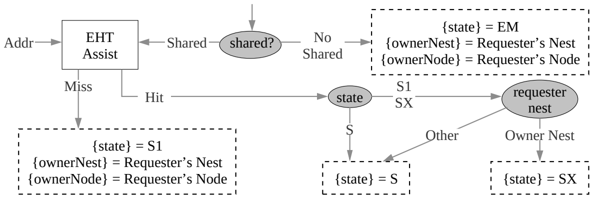


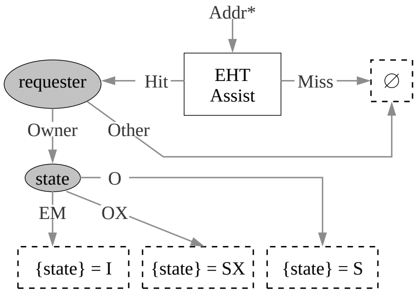
Figure 6: Updating the EHTA by DRs.

situation, if the load requester is local to the home nest, neither the state nor the owner fields will be modified because the copies cached outside the home nest does not change. If the requester is located in the owner nest and the state field is either EM or OX, all the copies outside the home nest will be located in the same nest and, consequently, the state field is set to OX. In case the requester is not located in neither the home nest nor the owner nest, the state field transitions to O. When the state field is set to O, DPs do not change the state.

In case of a store transaction, the EMC^2 chip will receive a DP only when there exists just a single copy of the block outside the home nest which, in addition, will be the owner (EM state). In such a case, if the requester is remote to the home nest, the state field transitions to EM and the owner field is set to the requester node (onwerNest and ownerNode). Otherwise, if the requester node is local to the home nest, the EHTA entry is deallocated because the single copy outside the home nest will be invalidated and forwarded to the requester.

3.2.3. Directed Responses. Figure 6 shows how the EHTA is updated on a Directed Response (DR) receipt. Notice that if an EHTA is updated by using a DR, it means that the owner is located inside the home nest and that the request is located outside. If the DR conveys an exclusive copy of a memory block (this can be known by using the shared bit present in the header of the DRs), the state field in the EHTA transitions to EM and the owner field is set to the requester node. In case the DR conveys a shared copy, the state for that block is checked in the EHTA. If the state is I or an EHTA miss occurs, a new entry is allocated setting the state to S1 and keeping the node information in the *ownerNest* and *ownerNode* fields. If an EHTA hit happens, the EHTA state is S1 or SX, and the requester nest matches the *ownerNest* field, the state is set to SX. If the state is S1 or SX and the requester nest does not matches the *ownerNest* field, the state is updated to S. If the state is S, it is kept.

3.2.4. Target Done Messages. A Target Done (TD) packet can only cause the EHTA to be updated when it has been generated as a result of a cache replacement (VicBlk transaction). Notice that, only the block's owner can initiate a replacement because the shared copies are replaced silently. Upon the arrival of a TD, if the owner in the EHTA coincides with the requester of the VicBlk transaction, the state field is checked. If the state is EM, it transitions to I because the single copy outside the home nest has been invalidated. If the state is OX, it transitions to SX because the copy held by the owner is invalidated, but the remaining copies may continue to be valid. Finally, if the state is O, it transitions to S because the other copies in the system may continue to be valid. On the contrary, if the owner in the EHTA does not coincide with the requester, the EHTA is not modified because a race condition happened and the EHTA has already been correctly updated. These operations are depicted in Figure 7.



* Retrieved from the MST

Figure 7: Updating the EHTA by local TDs.

4. Scalability Issues

This section addresses two potential scalability problems that could appear in the proposed extension of the coherence protocol. They come as a consequence of constraints imposed by Magny-Cours. The first one is the limited size of the HTA structure and it affects the HTA coverage ratio. The second one, is the limited number of tags (32) available to translate external transactions into internal transactions. In the next sections, we describe these issues in more detail and propose two mechanisms to prevent them from being a bottle-neck in large-scale configurations.

4.1. HTA Coverage Ratio

As previously described in Section 2, each HTA keeps coherence information of its local blocks. The number of entries of each HTA (256K) is twice the number of entries of the cache hierarchy of each die (128K). Therefore, as stated in [4], the typical coverage ratio of the HTA is $\times 2$.

This ratio assumes a uniform distribution of lines among HTAs. However, considering that memory is not interleaved among the different dies, it is quite probable that some memory controllers will hold more cached blocks than others. The worst case scenario will appear when all the cached blocks belong to the same memory controller. In Magny-Cours, having up to 8 dies, the coverage ratio in that worst case decreases down to $\times 0.25$, which could be reasonable.

Unfortunately, when we move to larger systems, this worst-case coverage ratio can fall drastically (down to $\times 0.062$ for 32 dies), which could result in a significant number of cache invalidations due to HTA replacements. These invalidations would increase the L3 cache miss rate, which may lead to a significant performance degradation.

A solution for this problem is to interleave memory blocks or memory pages inside each nest (i.e., among the dies belonging to the same nest), but not to interleave among nests. The intra-nest interleaving lessens the impact of the worst-case coverage ratio by homogeneously distributing blocks among HTAs in the same nest.

On the other hand, we do not perform an inter-nest interleaving to avoid a performance degradation, mainly for sequential applications. Notice that it would be desirable for such applications to access as much as possible the local memory bank. Otherwise, they may significantly increase their average memory access latency. Since accessing to a memory bank in a local nest is much less expensive than the access to a memory bank in a remote nest (inter-nest communication has larger latency than intra-nest communication), this approach offers a very good trade-off between coverage ratio and access latency.

4.2. Matching Store Table Tags Limitation

As previously commented, in Magny-Cours there are only 32 tags per die available. Therefore, the Matching Store Table (MST) only has 32 entries for the incoming messages belonging to external transactions. When all entries in the MST are occupied, subsequent incoming messages must wait at the Pending Command Queue until an entry is deallocated. Again, as the system size grows, this limitation may become a bottle-neck and may degrade performance.

Since our system configuration has four dies per nest and one bridge chip, there are three unused die identifiers in the nest. Therefore, we propose to increase the number of available tags for the incoming messages belonging to external transactions. This would allow us to assign up to 96 additional tags (128 tag in total). In this way, the number of in-progress coherence messages belonging to external transactions also increases.

5. Simulation Environment

We evaluate our proposals with full-system simulation using Virtutech Simics [12] extended with the Wisconsin GEMS toolset [13], which enables detailed simulation of multiprocessor systems. For modeling the interconnection network, we have used GARNET [14], a detailed network simulator included in GEMS. Finally, we have also used the CACTI 5.3 tool [15], assuming a 45nm process technology, to measure the area required by our proposals.

Table 3: System parameters.

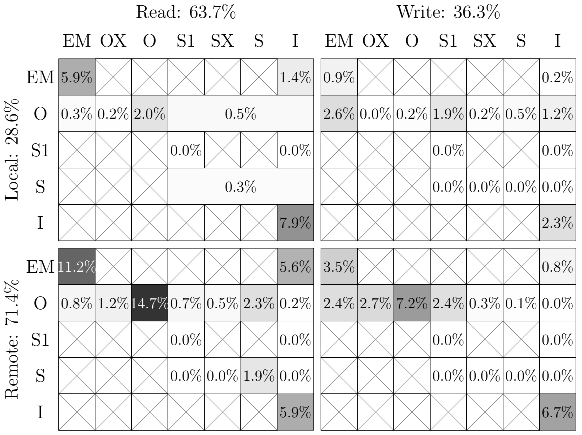
Memory Parameters	
Processor frequency	3.2 GHz
Cache block size	64 bytes
Aggregate L1+L2 caches	3MB, 4-way
L3 cache	5MB, 16-way
Average cache access latency (L1+L2+L3)	2ns
HT assist (probe filter)	1MB, 4-way
HT assist access latency	4ns
EMC^2 chip processing latency	16ns
Memory access latency (local bank)	100ns
Network Parameters	
Intra-nest topology	Fully-connected
Inter-nest topology	Hypercube
Data message size	68 or 72 bytes
Control message size	4 or 8 bytes
HyperTransport bandwidth (16 bits, 6.4GT/s)	12.8GB/s
Inter-die link latency	2ns
Inter-socket link latency	20ns
InfiniBand bandwidth (12x, 10Gb/s)	12GB/s
Inter-nest communication (one way)	150ns
Flit size	4 bytes
Link bandwidth	1 flit/cycle

For the evaluation of our proposals, we have first implemented the Magny-Cours cache coherence protocol. Then we have designed and implemented the behavior and the architecture of three different EMC^2 chips explained in Section 3. We have also provided the simulator with the functionality of having several cores per die sharing the same L3 cache. However, the intra-die coherence has not been modeled since (1) it is out of the scope of work and (2) the simulation time would increase considerably. We have run simulations from 8 to 32 dies and with 1 and 2 cores per die. For the Magny-Cours (*MC*) system we only simulate one nest with 8 dies. For the EMC^2 system we simulate 4 dies per nest (plus the EMC^2 chip). The parameters assumed for the systems evaluated in this work are shown in Table 3. Since we do not model the intra die protocol or the cache hierarchy, we assume a fixed access latency (representing the average access time) for the whole hierarchy (L1, L2, and L3 caches).

We have evaluated our proposal with a wide variety of scientific workloads from the SPLASH-2 benchmark suite [16]: *Barnes* (16K particles), *Cholesky* (tk16), *FMM* (16K particles), *Ocean* (514×514 ocean) *Raytrace* (teapot) and *Water-Sp* (512 molecules). All the experimental results reported in this work correspond to the parallel phase of these benchmarks. We account for the variability in multithreaded workloads [17] by doing multiple simulation runs for each benchmark in each configuration and injecting small random perturbations in the timing of the memory system for each run.

6. Evaluation Results

In this section, we show how our proposals support more than 8 dies while scaling in terms of execution



		Read: 63.7%							Write: 36.3%						
		EM	OX	O	S1	SX	S	I	EM	OX	O	S1	SX	S	I
Local: 28.6%	EM	5.9%	☒	☒	☒	☒	☒	1.4%	0.9%	☒	☒	☒	☒	☒	0.2%
	O	0.3%	0.2%	2.0%	0.5%				2.6%	0.0%	0.2%	1.9%	0.2%	0.5%	1.2%
	S1	☒	☒	☒	0.0%	☒	☒	0.0%	☒	☒	☒	0.0%	☒	☒	0.0%
	S	☒	☒	☒	0.3%				☒	☒	☒	0.0%	0.0%	0.0%	0.0%
	I	☒	☒	☒	☒	☒	☒	7.9%	☒	☒	☒	☒	☒	☒	2.3%
Remote: 71.4%	EM	11.2%	☒	☒	☒	☒	☒	5.6%	3.5%	☒	☒	☒	☒	☒	0.8%
	O	0.8%	1.2%	14.7%	0.7%	0.5%	2.3%	0.2%	2.4%	2.7%	7.2%	2.4%	0.3%	0.1%	0.0%
	S1	☒	☒	☒	0.0%	☒	☒	0.0%	☒	☒	☒	0.0%	☒	☒	0.0%
	S	☒	☒	☒	0.0%	0.0%	1.9%	0.0%	☒	☒	☒	0.0%	0.0%	0.0%	0.0%
	I	☒	☒	☒	☒	☒	☒	5.9%	☒	☒	☒	☒	☒	☒	6.7%

Figure 8: Characterization of cache misses according to the HTA (vertical) and EHTA (horizontal) states, read/write misses, and local/remote misses. Results show the average of all the evaluated benchmarks. Crossed cells represent impossible combinations of states. The darker the color of a cell is, the higher the miss percentage is. Larger cells indicate that the EHTA is not reached, and therefore, the state can be any one of those covered by the cell.

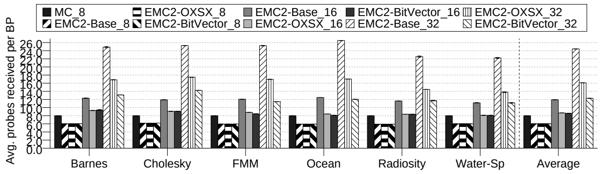


Figure 9: Number of probes received for each broadcast probe sent by the home die.

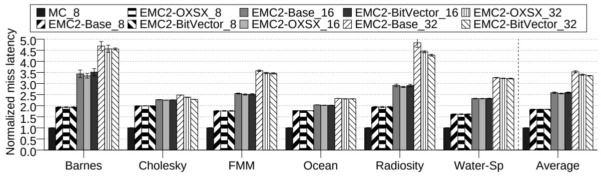


Figure 10: Normalized miss latency for the evaluated configurations.

time. Additionally, we compare the three bridge chips proposed in this paper in terms of network traffic, cache miss latency, execution time, and area requirements. Finally, we study the impact of the mechanism proposed in Section 4.1 to increase the HTA coverage ratio. Since the number of nodes of the evaluated configurations is not enough to fill the MST, we do not perform an evaluation of a system that employs the unused die identifiers in the nest to increase the number of available tags since it is not necessary.

6.1. Cache Miss Characterization

First of all, it is important to characterize the applications in order to get an idea of the percentage of cache misses that can take advantage of the EHTA filtering capabilities. Figure 8 shows this characterization for a 32-die system with the *EMC*^{2}*-OXSX* chip, as a representative example (see Table 2).

Our EMC^2 chips can reduce network traffic only when a write miss happens for a block in O or S states in the HTA (i.e., when a Broadcast Probe is received). On average for the considered applications, this happens for 21.7% of cache misses in a 32-die configuration. Depending on the state in the EHTA, the EMC^2 chip can either filter the Broadcast Probe or convert it into a Directed Probe. Note that for the remaining misses, the HTA already filters the probes.

6.2. Network Traffic

For each Broadcast Probe issued by the home node, we show in Figure 9 the average number of Broadcast/Directed Probes that arrive to the dies. This number is plotted for systems from 8 to 32 dies, with one core per die, and for the three EMC^2 chips proposed and the base Magny-Cours system with 8 dies. Without any filtering this number should be 8, 16, and 32 for 8-, 16-, and 32-die systems, respectively.

Since Magny-Cours does not filter Broadcast Probes (due to write misses), the average number of probes arriving to a die is always 8. However, for the same system size our protocols reduce this number by filtering some probes. Obviously, when we consider 8 dies (i.e., 2 nests), there is only one remote nest, so all EMC^2 chips behave in the same way. For larger systems, we can see that the more coherence information the HTA stores, the more traffic it filters. Particularly, for a 32-die system we can see that the average number of received probes is reduced by 23.6% (24.4/32), 49.7% (16.1/32), and 61.6% (12.3/32) for *EMC*^{2}*-Base*, *EMC*^{2}*-OXSX*, and *EMC*^{2}*-BitVector*, respectively.

This reduction in the number of probes received by the dies has two consequences: (1) the number of generated probe responses is also reduced, and (2) the network congestion and the coherence controller congestion decreases. They lead to less time waiting for Probe Responses, and therefore, shorter cache miss latency, which will results in improvements in execution time.

6.3. Execution Time

As we can see in Figure 10, cache miss latency increases when we move from *MC_8* to *EMC*^{2}*_8*. This is because the latency for transmitting messages between nests is higher than between dies. Remember that in *MC_8* we have the 8 dies in the same nest, while in *EMC*^{2}*_8* the 8 dies are distributed in two nests.

On the other hand, when we consider a larger system, the cache miss latency increases. Nevertheless, we reduce the final execution time because the

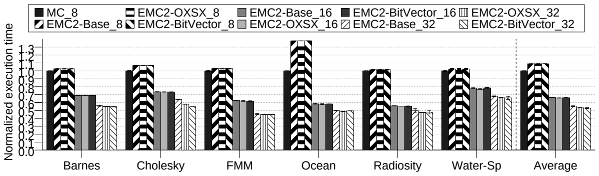


Figure 11: Normalized execution time for the evaluated configurations.

applications can be distributed among more dies which considerably reduces the workload of each die. Finally, we can observe the reduction in average cache miss latency achieved for some EMC^2 chips for the 32die configuration. Compared to *EMC^2-Base*, *EMC^2-OXSX* reduces the average miss latency by 3.7%, and *EMC^2-BitVector* by 5.0%. As we can observe, this percentage is expected to increase for larger scale configurations. These reductions in cache miss latency finally translate into improvements in execution time.

Figure 11 shows the normalized execution time when we increase the number of dies. We can see that, although for the 8-die configuration our proposals behave worse than *MC_8* (due to the inter-nest latency), when we extend the coherence domain through the bridge chip and allow a higher number of nodes, the execution time of the applications is significantly reduced. Particularly, *EMC^2-OXSX_32* and *EMC^2-BitVector_32* improve *MC* by 47% on average.

Comparing the three proposals in a 32-die system, *EMC^2-OXSX* and *EMC^2-BitVector* obtain similar execution time and slightly improve *EMC^2-Base* (≈4%).

6.4. Area Requirements

The different EMC^2 chips cover a wide trade-off between memory requirements and filtered traffic. This section studies the area of these chips and their trade-offs for a 32-die configuration.

The three chips differ in the size of the EHTA. Its sizes and those of the ETT and MST are described in Table 4. The EHTA of the *EMC^2-Base* is the one that less bits needs per entry (the tag plus 8 bits including state, owner die, and owner nest). The EHTA of the *EMC^2-OXSX* needs an extra bit for codifying the two additional states. Finally, the EHTA of the *EMC^2-BitVector* needs seven extra bits for storing the vector of remote nests.

Figure 12 plots the trade-off of these three chips in terms of network traffic and area requirements. The total area of each chip has been calculated by adding the areas (in mm^2) of the three data structures presented in the chip (without considering control logic). The normalized network traffic corresponds to the average number of flits transmitted by each switch in the whole system for the six benchmarks evaluated in this work, and normalized to *EMC^2-Base*.

Table 4: Size of the different bridge chips for systems up to 32 dies (8 nests).

	Structure	Entries	Assoc	Entry size	Area
	ETT	128	1	540 bits	0.64mm^2
	MST	32	1	607 bits	0.23mm^2
EMC^2-Base	**EHTA**	1M	16	tag + 8 bits	25.72mm^2
EMC^2-OXSX	**EHTA**	1M	16	tag + 9 bits	25.97mm^2
EMC^2-BitVector	**EHTA**	1M	16	tag + 15 bits	33.38mm^2

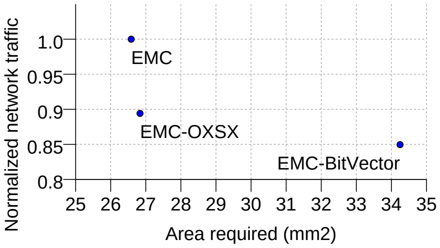


Figure 12: Traffic-area trade-off for a 32-die system.

We can observe that, *EMC^2-OXSX* reduces the traffic by 10.6% compared to *EMC^2-Base*, while *EMC^2-BitVector* reduces the traffic by 15%. Moreover, the area of *EMC^2-OXSX* is very close to the area of *EMC^2-Base*. Therefore, we can conclude that *EMC^2-OXSX* achieves a good compromise between network traffic and area requirements.

6.5. HTA Coverage Ratio

As discussed in Section 4, the HTA coverage ratio can be a problem for large scale systems, when the worst case scenario appears (i.e., most cached blocks map to the same home). In order to show this problem we have run simulations with two cores per die, four dies per nest, and two nests. Additionally, since the working set of the splash-2 benchmarks is very small compared to the cache sizes, we have halved the size of the caches (data and HTA) to be able to produce a meaningful evaluation while keeping the simulation time manageable. Note that halving both caches keeps the HTA coverage ratio constant.

In order to see the effect of the coverage ratio, we have split cache misses into a new 5C classification: the traditional 3C classification (*Cold* misses, *Capacity* misses and *Conflict* misses), *Coherence* misses, and *Coverage* misses. Coverage misses are caused by previous probe messages issued by the home node due to replacements in the HTA, which entails to invalidate all the copies of the block cached in the system. As we can see in Figure 13, when we do not perform any interleaving of memory addresses, the HTA can cause up to 50% of cache misses (e.g., in *Ocean*), and 25% on average. However, by using the hybrid interleaving policy described in Section 4, this percentage of misses is reduced significantly (up to 3%). This reduction in the number of cache misses (by 20%, on average)

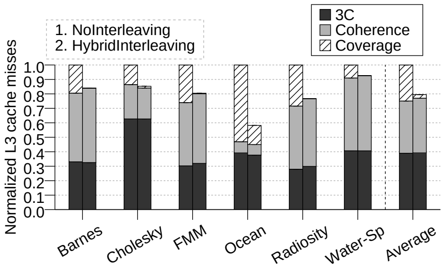


Figure 13: Classification of cache misses for the no-interleaving policy and the hybrid interleaving policy.

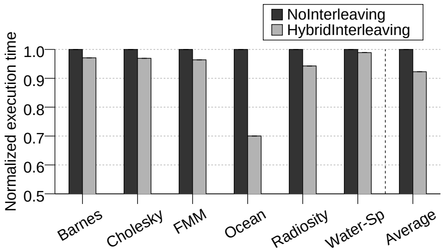


Figure 14: Normalized execution time for the no-interleaving policy and the hybrid interleaving policy.

results in improvements in execution time. We can observe in Figure 14 that execution time is reduced with the proposed policy by 7.8% on average.

7. Conclusions

In this paper, we have extended by an external logic (EMC^2 chip) the coherence domain of the AMD Magny-Cours processors beyond the 8-die limit imposed by both the cHT specification and the owner field of the HTA. The proposed chip not only maintains the HTA capability to filter the coherence traffic over the entire system, but also filters additional traffic, which provides the scalability required to build large-scale servers. Evaluation results for up to a 32-node system show how the runtime of the applications scales with the number of nodes, reducing the application runtime by 47% on average (compared to the 8-die Magny-Cours system).

We have proposed and analyzed three EMC^2 chip configurations able to provide different tradeoffs between filtered network traffic and required silicon area. Particularly in a 32-die system, *EMC^2-OXSX* achieves a good compromise between network traffic (10.6% of traffic reduction compared to *EMC^2-Base*) and reducing area requirements (22.2% of area reduction compared to *EMC^2-BitVector*).

In addition, we have also addressed two potential scalability problems that could degrade the performance of (very) large systems. In particular, the HTA coverage ratio problem can be palliated by a hybrid interleaving policy, reducing execution time by 7.8%.

Acknowledgment

This work has been supported by Generalitat Valenciana under Grant PROMETEO/2008/060, by Spanish Ministry of Ciencia e Innovación under grant “TIN2006-15516-C04-03”, and by European Comission FEDER funds under grant “Consolider Ingenio-2010 CSD2006-00046”. Antonio Robles is taking a sabbatical granted by the Universidad Politcnica de Valencia for updating his teaching and research activities.

References

[1] J. M. Owen, M. D. Hummel, D. R. Meyer, and J. B. Keller, “System and method of maintaining coherency in a distributed communication system,” U.S. Patent 7069361, Jun. 2006.

[2] Intel, “An introduction to the Intel QuickPath interconnect,” whitepaper, Jan. 2009. [Online]. Available: http://www.intel.com/technology/quickpath/introduction.pdf

[3] *InfiniBand Architecture specification release 1.2*, InfiniBand Trade Association$^{\text{TM}}$, Oct. 2004. [Online]. Available: http://www.InfiniBandta.com

[4] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, “Cache hierarchy and memory subsystem of the AMD opteron processor,” *IEEE Micro*, vol. 30, no. 2, pp. 16–29, Apr. 2010.

[5] P. Conway, “Computer system with integrated directory and processor cache,” U.S. Patent 6868485, Mar. 2005.

[6] SGI, “Technical advances in the SGI Altix UV architecture,” whitepaper, 2009. [Online]. Available: http://www.sgi.com/pdfs/4192.pdf

[7] 3Leaf Systems, “Next generation hybrid systems for HPC,” whitepaper, 2009. [Online]. Available: http://www.3leafsystems.com/download/3leaf_wt_paper_Next_Gen_Hybrid_Sys%tems_for_HPC.pdf

[8] J. Duato, F. Silla, S. Yalamanchili, B. Holden, P. Miranda, J. Underhill, M. Cavalli, and U. Brning, “Extending HyperTransport protocol for improved scalability,” in *1st Int’l Workshop on HyperTransport Research and Applications (WHTRA)*, Feb. 2009, pp. 46–53.

[9] R. Kota and R. Oehler, “Horus: Large-scale symmetric multiprocessing for opteron systems,” *IEEE Micro*, vol. 25, no. 2, pp. 30–40, Mar. 2005.

[10] A. Agarwal, R. Simoni, J. L. Hennessy, and M. A. Horowitz, “An evaluation of directory schemes for cache coherence,” in *15th Int’l Symp. on Computer Architecture (ISCA)*, May 1988, pp. 280–289.

[11] P. Sweazey and A. J. Smith, “A class of compatible cache consistency protocols and their support by the IEEE futurebus,” in *13th Int’l Symp. on Computer Architecture (ISCA)*, Jun. 1986, pp. 414–423.

[12] P. S. Magnusson, et al, “Simics: A full system simulation platform,” *IEEE Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[13] M. M. Martin, et al, “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset,” *Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Sep. 2005.

[14] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, “GARNET: A detailed on-chip network model inside a full-system simulator,” in *IEEE Int’l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2009, pp. 33–42.

[15] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, “Cacti 5.1,” HP Labs, Tech. Rep. HPL-2008-20, Apr. 2008.

[16] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in *22nd Int’l Symp. on Computer Architecture (ISCA)*, Jun. 1995, pp. 24–36.

[17] A. R. Alameldeen and D. A. Wood, “Variability in architectural simulations of multi-threaded workloads,” in *9th Int’l Symp. on High-Performance Computer Architecture (HPCA)*, Feb. 2003, pp. 7–18.