An Architecture for High-Performance Scalable Shared-Memory Multiprocessors Exploiting On-Chip Integration

Manuel E. Acacio, José González, *Member*, *IEEE Computer Society*, José M. García, *Member*, *IEEE*, and José Duato, *Member*, *IEEE*

Abstract—Recent technology improvements allow multiprocessor designers to put some key components inside the processor chip, such as the memory controller, the coherence hardware, and the network interface/router. In this paper, we exploit such integration scale, presenting a novel node architecture aimed at reducing the long L2 miss latencies and the memory overhead of using directories that characterize cc-NUMA machines and limit their scalability. Our proposal replaces the traditional directory with a novel three-level directory architecture, as well as it adds a small shared data cache to each of the nodes of a multiprocessor system. Due to their small size, the first-level directory and the shared data cache are integrated into the processor chip in every node, which enhances performance by saving accesses to the slower main memory. Scalability is guaranteed by having the second and third-level directories out of the processor chip and using compressed data structures. A taxonomy of the L2 misses, according to the actions performed by the directory to satisfy them, is also presented. Using execution-driven simulations, we show that significant latency reductions can be obtained by using the proposed node architecture, which translates into reductions of more than 30 percent in several cases in the application execution time.

Index Terms—cc-NUMA multiprocessor, directory memory overhead, L2 miss latency, three-level directory, shared data cache, on-processor-chip integration.

1 Introduction

Although cc-NUMA constitutes an attractive architecture for building high-performance shared-memory multiprocessors, far away from the limits imposed by snooping protocols, there are a number of factors that constrain the amount of nodes that can be offered at a good price/performance ratio. Two of these factors are the increased cost in terms of hardware overhead and the long L2 miss latencies.

An important component of the hardware overhead is the amount of memory required to store directory information, which depends, to a large extent, on the type of the sharing code used in each one of the entries of the directory. On one hand, using a *full-map* (or bit-vector) sharing code leads to prohibitively high memory requirements for large-scale configurations of a cc-NUMA multiprocessor. On the other hand, compressed sharing codes (such as *coarse-vector*) significantly reduce directory memory overhead by storing an in-excess codification of the sharers (i.e., more sharers than necessary are usually included). Unfortunately, this compression has negative consequences on performance, which are motivated by the *unnecessary coherence messages* they introduce. Two-level directories have been recently proposed as an effective means of significantly reducing directory memory overhead while keeping performance [1].

The second of the previous factors (i.e., the long L2 miss latencies suffered in cc-NUMA designs) is caused by the inefficiencies that the distributed nature of the protocols and the underlying scalable network imply. One of such inefficiencies is the indirection introduced by the access to the directory information, which is usually stored in main memory.

L2 misses can be classified according to the actions performed by directories in order to resolve them. This way, we can distinguish roughly two kinds of L2 misses: those for which the corresponding home directory has to provide the memory line and those for which the memory line is not sent by the home directory. For those misses belonging to the latter category, the access to main memory in order to obtain directory information has a serious impact that significantly increases their latency when compared to symmetric multiprocessors (SMPs). The fact that these L2 misses appear very frequently [2] along with the well-known industry trend that microprocessor speed is increasing much faster than memory speed (the *memory wall* problem) [3] have motivated that many efforts are devoted to extend the scale at which snoopy coherence protocols can be applied. The Sun Enterprise E10000 [4] constitutes an example of a recent SMP which can accommodate up to 64 processors.

- *M.E. Acacio and J.M. García are with the Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, Campus de Espinardo S/N, Facultad de Informática, 30071 Murcia, Spain. E-mail: {meacacio, jmgarcia}@ditec.um.es.*
- *J. González is with the Intel Barcelona Research Center, Intel Labs Barcelona, C/ Jordi Girona 29, Edif. Nexus 2, Planta 3, 08034 Barcelona, Spain. E-mail: pepe.gonzalez@intel.com.*
- *J. Duato is with the Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia, Camino de Vera S/N, 46010 Valencia, Spain. E-mail: jduato@gap.upv.es.*

Manuscript received 13 Mar. 2003; revised 20 Oct. 2003; accepted 3 Nov. 2003.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0031-0303.

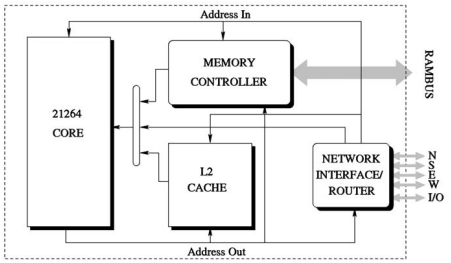

Fig. 1. Alpha 21364 EV7 diagram.

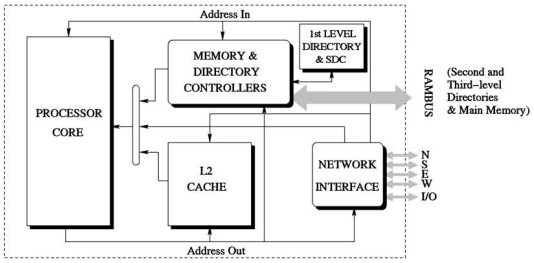

Fig. 2. Proposed chip organization.

On the other hand, advances in VLSI have enabled a computing model where many processors will soon be integrated into a single die with little impact on system cost. Besides, technology improvements have already allowed the integration of some key components of the system inside the processor chip. For example, the Compaq Alpha 21364 EV7 [5] includes on-chip memory controller, coherence hardware, and network interface and router (see Fig. 1).

We take advantage of the opportunities provided by current integration scale presenting a novel node organization especially designed to reduce the latency of L2 misses and the memory overhead of using directories that characterize cc-NUMA multiprocessors and limit their scalability.

We extend the two-level directory architecture previously presented in [1] by adding another directory level which is included into the processor chip (along with the coherence controller). This results in a three-level directory architecture. The new directory level becomes the first-level directory, whereas the small full-map directory cache and the compressed directory, which are placed outside the processor, are now the second and third-level directories, respectively. In addition, our proposal adds a small shared data cache to each one of the nodes forming the multiprocessor. The shared data cache (SDC) is also included into the processor die and contains those memory lines for which the home node holds a valid copy in main memory, and that are expected to be requested in the near future. Unlike the remote data caches (RDCs) used in some systems (such as NUMA-Q [6]) to cache lines that are fetched to the local node from remote memories, the shared data cache proposed in our design tries to reduce the latency of accessing main memory (when a copy of the memory line must be sent) by quickly providing data from the processor itself to the requesting node. This is possible since coherence hardware, memory controller, and network router are also included into the processor die. Fig. 2 illustrates the chip organization we propose in this paper.

The on-chip integration of the small first-level directory and the shared data cache enhances performance. Now, those cache misses that find their corresponding directory entry and memory line (when needed) in the first-level directory and shared data cache, respectively, can be directly served from the processor chip, significantly reducing the directory component of the L2 miss latency (that is, the time needed by the home directory to satisfy L2 misses). Additionally, organizing the directory as a multilevel-level structure significantly reduces the memory overhead of using directories. In this case, this is achieved by having two directory levels out of the processor chip and using compressed data structures.

The main contribution of this paper is the significant reduction in the L2 miss latency (more than 60 percent in some cases), which leads to performance improvements up to 50 percent over a traditional cc-NUMA multiprocessor using a full-map and with the coherence hardware included on-chip. The simplicity of our proposal and the fact that it could be introduced on commercial processors cuts its cost off, conversely to the expensive sophisticated network designs required by state-of-the-art moderate-scale SMPs.

This paper extends the work previously presented in [7], including a more detailed description and evaluation of our proposal. The rest of the paper is organized as follows: The related work is presented in Section 2. Section 3 proposes a taxonomy of the L2 misses found in cc-NUMA multiprocessors in terms of the actions executed by directories. The new node architecture is proposed and justified in Section 4. Section 5 discusses our evaluation methodology. Section 6 shows a detailed performance evaluation of our novel proposal and, finally, Section 7 concludes the paper.

2 Related Work

System-on-a-chip techniques allow for integration of all system functions including compute processor, caches, communications processor, interconnection networks, and coherence hardware onto a single die. Directly connecting these highly-integrated nodes leads to a high-bandwidth, low-cost, low-latency, "glueless" interconnect. Some proposals exploiting system-on-a-chip techniques are the Compaq Alpha 21364 EV7 [5], the IBM BlueGene/L supercomputer [8], and the AMD Hammer [9]. Even more, as more transistors can be placed on a chip, a sizeable fraction of the main memory is likely to be also integrated on chip (processor-in-memory or PIM chips) [10]. Other designs go further and use semiconductor technology trends to implement a complete multiprocessor into a single chip (multiprocessor-on-a-chip), for example, the Compaq Piranha CMP [11], the Stanford Hydra [12], or the IBM POWER4 [13]. In this work, we take advantage of the increased transistor counts by integrating a small first-level directory and shared data cache on chip, which results in significant reductions on the latency of L2 misses.

Some hardware optimizations, proposed to shorten the time processors loose because of cache misses and invalidations, were evaluated in [14]. In [15], the remote memory access latency is reduced by placing caches in the crossbar switches of the interconnect to capture and store shared data as they flow from the memory module to the requesting

processor. Subsequently, in [16], the same idea is applied to reduce the latency of cache-to-cache transfer misses. In both cases, special network topologies are needed to keep coherent the information stored in these switch caches.

Other proposals have focused on using snooping protocols with unordered networks [2], or using prediction-based techniques to reduce the latency of certain L2 misses (for example, [17], [18], [19], and [20]). In addition, remote data caches (RDCs) have also been used in several designs (as [6], [21], and [22]) to accelerate access to remote data. A RDC is used to hold those lines that are fetched to the node from remote memory and acts as backup for the processor caches.

Caching directory information (*sparse directories*) was originally proposed in [23] and [24] as a means to reduce the memory overhead entailed by directories. More recently, directory caches have also been used to reduce directory access times [25], [26]. The Everest architecture proposed in [27] uses directory caches to reduce directory access time. Everest is an architecture for high-performance cache coherence and message passing in partitionable distributed shared memory systems that use commodity SMPs as building blocks. In order to maintain cache coherence between shared caches included into every SMP, Everest uses a new directory design called Complete and Concise Remote (CCR) directory. In this design, each directory maintains a *shadow* of the tags array of each remote shared cache. In this way, each directory requires $N-1$ shadows for a N-node system, which prevents CCR directories from being a solution for scalable systems.

Some authors propose to reduce the width of directory entries by having a limited number of pointers per entry to keep track of sharers [28], [29], [30], or by using a compressed sharing code, such as *coarse vector* [23] or *BT-SuT* [1].

The use of sparse directories tends to increase the number of cache misses as a result of the *unnecessary invalidations* that are sent each time a directory entry is evicted. On the other hand, compressed sharing codes and some of the limited pointers schemes (as a consequence of directory overflows) increase the number of messages that are sent on every coherence event due to the appearance of *unnecessary coherence messages*. Unfortunately, unnecessary invalidations and unnecessary coherence messages can significantly degrade performance. In order to significantly reduce the memory overhead entailed by the directory information while achieving the same performance as a nonscalable full-map directory, we proposed in a previous work a two-level directory architecture, which combined a small first-level directory (a few full-map entries) with a compressed and complete second level. The aim of such organization is to provide precise information for those memory lines that are frequently accessed and in-excess information for those lines that are rarely accessed [1].

3 Taxonomy of L2 Misses

Each time an access to a certain memory line from a particular processor (i.e., a load or a store) misses in the L2 cache, a request for the line is addressed to the corresponding *home* directory. The home directory accesses directory information in order to obtain both the state of the line and a list of those nodes (if any) that hold a copy of the memory line. Each memory line can be in one of the following three states (MESI states are used for L2 caches):

- *Uncached*: The memory line is not cached by any node and the home directory has a valid copy of the line.
- *Shared*: Several L2 caches are holding read-only copies of the memory line, and the home directory has a valid copy of the line.
- *Private*: There is a single L2 cache having a read-write copy of the memory line. Now, the copy stored by the home directory is not guaranteed to be valid and, thus, the copy hold by the owner node should be obtained.

Depending on both the state of the line and the type of access causing the miss, the directory determines the actions needed to satisfy the miss. We classify L2 misses in terms of these actions into four categories:

- ***$-to-$* Misses**: Cache-to-cache transfer misses occur when the requested line is in the *Private* state, that is, there is a single processor caching the memory line in the *Modified* or in the *Exclusive* states. In this case, the home directory forwards the miss to the current owner of the line. Then, the owner sends a reply with the line directly to the requestor and a revision message to the home directory. These misses are also known as *3-Hop misses*.
- ***Mem* Misses**: Memory misses appear for write accesses when there is no processor caching the requested line (the line is in the *Uncached* state), and for read accesses when the requested line is cached by zero or more than one processor (the state of the line is *Uncached* and *Shared*, respectively). The home directory has a valid copy of the memory line and satisfies the miss by accessing the main memory in order to directly provide the line.
- ***Inv* Misses**: Invalidation misses, also known as *upgrade misses*, take place when a write access for a memory line comes to the directory, there are several nodes holding a copy of the line (the line is in the *Shared* state) and one of them is the processor issuing the access (that is to say, this processor sent an *upgrade* for the line). In this situation, the directory sends invalidation messages to all sharers of the memory line except the requesting processor. Once all the invalidation acknowledgment messages have been received, the directory provides ownership of the line to the requestor.
- ***Inv+Mem* Misses**: Invalidation and access to memory misses are caused by a write access for which there are several nodes caching the line, but none of them is the one issuing the access. Now, the directory must first invalidate all copies of the line. Once it has obtained all the invalidation acknowledgments, it sends a reply with the memory line to the requesting processor. The difference between *Inv* and *Inv+Mem* misses is that, for the first category, the directory does not provide a copy of the memory line since the requestor already has a valid one.

Table 1 summarizes the actions that the directory performs to satisfy load and store misses. A load miss is either satisfied by a cache-to-cache transfer or an access to memory. On the other hand, any of the four actions could be used for store misses. Observe that *Mem* misses are a

TABLE 1
Directory Actions Performed to Satisfy Load and Store Misses

	Directory States			
	Private	**Shared**		**Uncached**
Load Miss	*\$-to-\$*	*Mem*		*Mem*
Store Miss	*\$-to-\$*	*Inv* (Upgrade)	*Inv+Mem*	*Mem*

consequence of cold, capacity, conflict, or coherence misses, whereas *\$-to-\$*, *Inv*, and *Inv+Mem* misses fall into the coherence misses category.

For each one of the former categories, Figs. 3, 4, 5, and 6 present the normalized average miss latency obtained when running several applications on the base system assumed in this work. Average miss latency is split into network latency, directory latency, and miscellaneous latency (buses, cache accesses, etc.). Further details about the evaluation methodology are included in Section 5. As can be observed, the most important fraction of the average miss latency is caused by the directory. This is true for all applications in the *Inv* and *Inv+Mem* cases, whereas only four applications found network latency to exceed directory one in the *Mem* and *\$-to-\$* categories. For *\$-to-\$* and *Mem* misses, directory latency is caused by the access to main memory to obtain the corresponding directory entry (for *Mem* misses, memory line lookup occurs in parallel with the access to the directory information, as in [31]). For *Inv* and *Inv+Mem* misses, directory latency comprises the cycles needed to obtain directory information and to create and send invalidation messages, as well as to receive the corresponding acknowledgments (again, for a *Inv+Mem* miss, the directory entry and the requested line are obtained at the same time). In our system, *Inv* and *Inv+Mem* misses are temporally stored in the home directory until all the responses to the invalidation messages have been received. In this way, from the perspective of *Inv* and *Inv+Mem* misses, the cycles needed to invalidate all the sharers (if any) account as directory latency since these cycles are actually spent in the directory.

4 A Novel Architecture to Reduce L2 Miss Latency

4.1 Node Architecture

The proposed node organization adds several elements to the basic node architecture shown in Fig. 1.

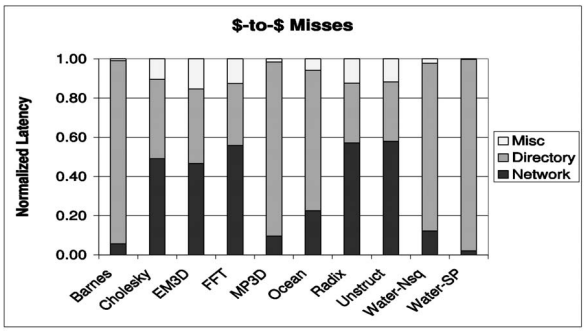

Fig. 3. Normalized average latency for *\$-to-\$* misses.

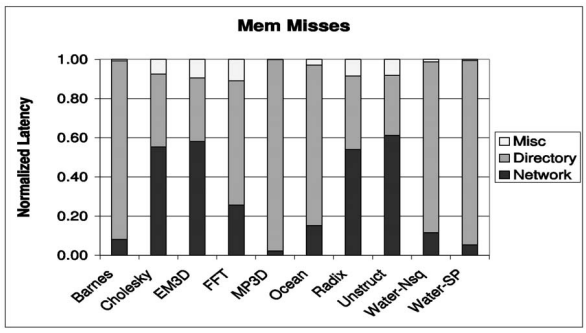

Fig. 4. Normalized average latency for *Mem* misses.

4.1.1 Three-Level Directory Architecture

Our proposal extends the two-level architecture presented in [1] adding a new directory level which is integrated into the processor chip. Two-level directories were originally proposed for reducing directory memory overhead while retaining performance. However, we can take advantage of their uncoupled nature and use them to also reduce L2 miss latency. It is important to note that coherence hardware, memory controller, and network router are already included inside the processor die.

The three-level directory architecture included in our proposal consists of:

1. *First-level directory*: This directory level is located inside the processor chip, close to the directory controller. It is managed as a cache and uses a small set of entries, each one containing a precise sharing code consisting of three pointers (of $\log_2 N$ bits each one, for an N-node system). Note that, as shown in [32], a small number of pointers generally is sufficient to keep track of the nodes caching a memory line. Besides, from the graphs presented in [33], one can conclude that in more than 90 percent of the cases, memory lines are shared among up to three sharers for most applications. Therefore, having three pointers per line is a good compromise between cost and performance for the scientific applications that have been used in these evaluations. On the other hand, choosing a sharing code linearly dependent on the number of processors (such as full-map) may make it infeasible to be incorporated on the processor chip, compromising both scalability and performance. In short, it is preferable to invest the transistor budget

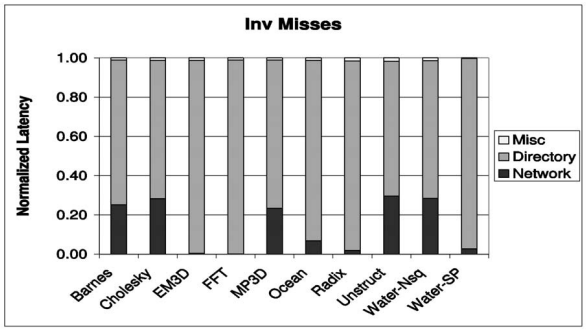

Fig. 5. Normalized average latency for *Inv* misses.

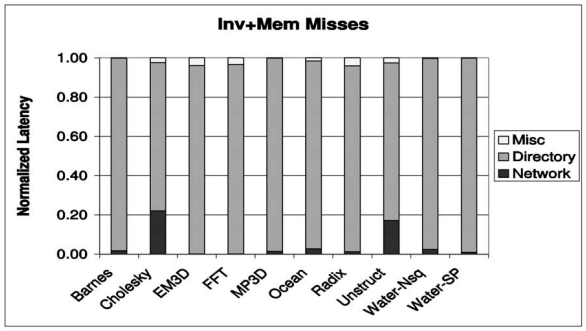

Fig. 6. Normalized average latency for *Inv+Mem* misses.

dedicated to this directory in increasing the number of entries (which increases the hit rate) rather than directory width.

2. *Second-level directory*: It is located outside the processor chip and also has a small number of entries. In this case, a nonscalable but precise full-map sharing code is employed. The second-level directory can be seen as a *victim cache* of the first level since it contains those entries that have been evicted from the first-level directory or do not fit there due to the number of sharers becoming larger than the available number of pointers (three in our case). It is important to note that, if a directory entry for a certain memory line is present at the first level, it cannot be present at the second, and vice versa.
3. *Third-level directory*: This level constitutes the complete directory structure (i.e., an entry per memory line) and is located near main memory (it could be included in main memory). Each entry in this level uses a compressed sharing code, in particular, the binary tree with subtrees (*BT-SuT*) sharing code (see [1] for details), which has space complexity O($log_2(N)$), for an *N*-node system. Sharing information in this level is always updated when changes in the first or second-level directories are performed.

Accesses to the third-level directory imply main memory latency. The first-level directory has the latency of a fast on-chip cache, whereas the second-level one provides data at the same speed as an off-chip cache. In this way, *$-to-$*, *Inv*, and *Inv+Mem* misses would be significantly accelerated if their corresponding directory entry were found at the first or second-level directories. In this case, the directory controller could immediately forward the miss to the owner of the line, in the case of an *$-to-$* miss, or send invalidation messages to the sharers of the line, in case of *Inv* and *Inv+Mem* misses. Note that, for *Inv+Mem* misses, the access to main memory would not be in the critical path of the miss since it would occur in parallel with the creation and sending of the invalidation messages. Due to the locality exhibited by memory references, we expect the first and second-level directories to satisfy most of the requests, even remote accesses to a home node. Thus, this will bring important reductions in the component of the miss latency owed to the directory.

4.1.2 Shared Data Cache (SDC)

Contrary to *$-to-$* misses, *Inv* misses and *Inv+Mem* misses which can significantly benefit from finding directory information in the first or second-level directories, *Mem* misses cannot take full advantage of having the corresponding directory entry in these directory levels. For these misses, the directory controller must directly provide the memory line. Therefore, main memory has to be accessed, being in the critical path of the miss.

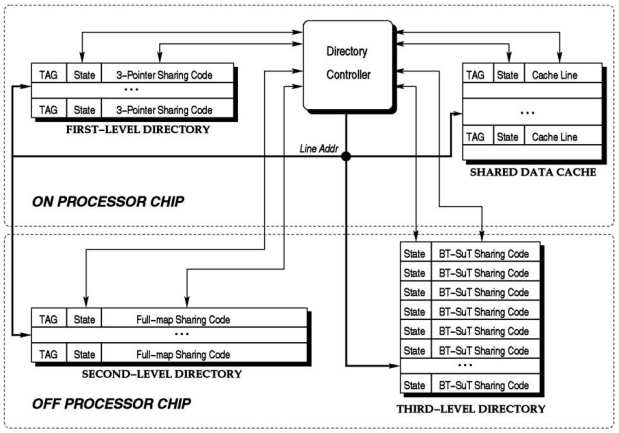

Fig. 7. Proposed node architecture.

In order to also accelerate *Mem* misses, our design includes a small cache inside the processor chip, the *shared data cache (SDC)*. This cache is used by the directory controller of each home node to store a copy of those memory lines for which a valid copy is present in main memory (that is to say, they are in the *Shared* or *Uncached* states) and that are expected to be accessed in a near future. As coherence hardware, memory controller, and network router are already included inside the processor chip, *Mem* misses can take significant advantage of finding their corresponding memory line in the shared data cache, since the access to the slower main memory would be avoided. State bits are also included in each entry of this shared data cache. The reason for this will be explained in next section.

Note that, unlike the remote data caches (RDCs) used in some systems to cache lines that are fetched to the node from remote memories, the shared data cache proposed in our design holds some of the memory lines that, having been assigned to the local node (i.e., it is the home node for that lines), are expected to be accessed in a near future, and valid copies of them are stored in main memory. Thus, this structure is used by the directory controller of the home node to reduce the latency of accessing main memory (when needed) by quickly providing data from the processor itself to the requesting node.

Fig. 7 summarizes the node architecture proposed in this paper. This node organization is divided into two different parts (on-chip and off-chip parts) according to their location in the processor chip. The on-chip part includes the first-level directory and the shared data cache. The off-chip structure comprises the second and third-level directories. Tag information is stored in the first and second-level directories as well as the shared data cache in order to determine whether there is a hit. The next section describes how the proposed node architecture operates to satisfy L2 misses.

4.2 Directory Controller Operation

Each time an L2 miss for a certain memory line reaches the directory controller, the address of the line associated with the request is sent to each one of the directory levels as well

as the shared data cache and main memory. One of the following situations will take place:

1. **The directory entry is found in the first-level directory**. In this case, the directory controller obtains precise sharing information, ending the access to the second and third levels. Additionally, for *$-to-$* and *Inv* misses, the accesses to the shared data cache and main memory are also cancelled since the home directory must not provide the memory line associated with the miss. For *Mem* and *Inv+Mem* misses, the access to main memory is ended if the memory line is found in the shared data cache. In all the cases, the directory controller properly updates sharing information in the first and third-level directories. After a *Mem* miss, it is possible that the final number of sharers will be greater than three. In this case, the number of pointers used in the first level is insufficient and, then, the entry in this level is freed and moved to the second level.
2. **The directory entry is found in the second-level directory**. Again, precise sharing information is obtained. The directory controller ends the access to the third-level directory whereas the access to the shared data cache has already been completed (due to the on-chip integration of the shared data cache, its latency is assumed to be lower than or equal to the one incurred by the second-level directory). Then, the access to main memory is cancelled if a *$-to-$* or an *Inv* miss is found or, for *Mem* and *Inv+Mem* misses, if the memory line was obtained from the shared data cache. Sharing information in the second and third levels is also updated and, if the final number of sharers is lower than four, the directory entry is moved from the second to the first-level directory.
3. **The directory entry is found in the third-level directory**. The third-level directory has a directory entry for each memory line, therefore it can provide sharing information for those memory lines for which an entry in the first and second-level directories has not been found. In this case, the directory controller obtains imprecise but correct sharing information which, once the miss is processed, is properly updated.

On an L2 miss for which an entry in the first-level directory was not found, an entry in this directory level is allocated only when *precise* sharing information can be guaranteed. This occurs in one of the following four cases:

1. The miss requested a private copy of the line. Once the miss has been completed, only the node that issued it caches a copy of the line.
2. The line was in the *Uncached* state. In this case, no sharing code was needed and the state of the line was obtained from the shared data cache (if the memory line was present in this structure) or from the third-level directory. Again, once the miss has been served, the line is only held by the requesting node.
3. An entry for the line was found in the second-level directory and the final number of sharers is not greater than the number of pointers used in the sharing code of the first-level directory (three in this design).
4. The line was in the *Private* state and the directory entry was provided by the third level. In this case, the sharing code used by the third-level directory (that is to say, *BT-SuT*) is wide enough to precisely codify the identity of the single sharer and, thus, precise sharing information is also available.

An entry in the shared data cache is allocated in one of these three situations:

1. On a *Mem* miss for which an entry is present in the first or second-level directories. In this case, there are several sharers for the line since otherwise, a directory entry in one of these levels would have not been used. Note also that the miss had to be caused by a load instruction (otherwise, an *Inv* or *Inv+Mem* miss would have been obtained).
2. On a write-back message from the owner node of the line. This message is caused by a replacement from the single L2 cache holding the memory line and always includes a copy of the line.[1] The line is inserted into the shared data cache (and also into main memory if it was modified). Since no sharing code is necessary, the state bits of the associated entry in the shared data cache are used to codify the state of the line (*Uncached* in this case). In this way, an entry in the first or second-level directories will not have been wasted.
3. On receiving the response to a *$-to-$* miss indicating that a cache-to-cache transfer was performed, when the miss was caused by a load instruction. In this case, the state of the line was changed from *Private* to *Shared*, and the acknowledgment message contains a copy of the line, which is included at this moment into the shared data cache (and also into main memory if it was modified).

An entry in the first and second-level directories is freed each time a write-back message for a memory line in the *Private* state is received. This means that this line is no longer cached in any of the system nodes, so its corresponding directory entry is made available for other lines. An entry in the shared data cache is freed in two cases: First, after a miss that obtains an exclusive copy of the memory line is processed. This takes place when either the miss was caused by a store instruction or the memory line was in the *Uncached* state (remember we are assuming a four-state MESI coherence protocol). And, second, when its associated directory entry is evicted from the second-level directory.

Replacements from the second-level directory are discarded since correct information is present in the third-level directory. The same occurs for those memory lines that are evicted from the shared data cache, since main memory contains a valid copy of them. Finally, replacements in the first and second-level directories are not allowed for the entries associated with those memory lines with pending coherence transactions. Table 2 summarizes the most important issues described in this section.

4.3 Implementation Issues

In this paper, we assume that the organization of the first and second-level directory caches as well as of the shared data cache is fully associative, with a LRU replacement policy. Practical implementations can be set-associative,

1. The original coherence protocol did not include a copy of the line neither in a write-back message nor in a cache-to-cache transfer response when the copy contained in main memory was valid.

TABLE 2
Directory Controller Operation

Directory Receives		Directory information found in: First-level directory (DC1)	Second-level directory (DC2)	Third-level directory (MEM)	Shared data cache (SDC)
L2 Miss	Miss Type				
	$-to-$	Update entry	Move entry to DC1	Allocate an entry in DC1	Not allowed
	Mem	Update entry (Sharers $\leq$ 3) or move entry to DC2 Insert in SDC (if needed)	Update entry (Sharers $>$ 3) or move entry to DC1 Insert in SDC (if needed)	Allocate an entry in DC1 (if line state = *Uncached*)	Allocate an entry in DC1 and extract line from SDC
	Inv	Update entry and extract line from SDC (if found)	Move entry to DC1 and extract line from SDC (if found)	Allocate an entry in DC1	Not allowed
	Inv+Mem				
Write-back		Free DC1 entry and insert line in SDC	Free DC2 entry and insert line in SDC	Insert line in SDC	Not allowed
$-to-$ Response		Insert line in SDC (if load miss)	Insert line in SDC (if load miss)	Nothing	Not allowed

achieving similar performance at lower cost [26]. Each line in the first and second-level directories contains a single directory entry. The state bits field used in every entry of the shared data cache are implemented using the already included state bits of each entry. Now, three states are possible for a line in the shared data cache: not present, present in the *Uncached* state, or present in the *Shared* state.

We assume that the directory controller provides write buffers to update the three directory levels, the shared data cache and main memory. Thus, writes to each one of the directory levels as well as to main memory and to the shared data cache are assumed to occur immediately.

Finally, the first and second-level directories are implemented as directory caches, using the usual technologies for on-chip and off-chip processor caches, respectively. The small sharing code used for the third-level directory would avoid the need of external storage for this directory level since, as in [11] and [22], it could be directly stored in main memory by computing ECC at a coarser granularity and utilizing the unused bits. This approach leads to lower cost by requiring fewer components and pins, and provides a simpler system scaling.

TABLE 3
Base System Parameters

64-Node System	
ILP Processor	
Processor Speed	1 GHz
Max. fetch/retire rate	4
Instruction Window	64
Functional Units	2 integer arithmetic 2 floating point 2 address generation
Memory queue size	32 entries
Cache Parameters	
Cache line size	64 bytes
L1 cache WT	Direct mapped, 32KB
L1 request ports	2
L1 hit time	2 cycles
L2 cache WB	4-way associative, 512KB
L2 request ports	1
L2 hit time	15 cycles, pipelined
Number of MSHRs	8 per cache
Directory Parameters	
Directory controller cycle (on-chip)	1 cycle
1^{st}-level directory access time (on-chip)	1 cycle
2^{nd}-level directory access time (off-chip)	10 cycles
3^{rd}-level directory access time (off-chip)	70 cycles
First coherence message creation time	4 cycles
Next coherence messages creation time	2 cycles
Memory Parameters	
Shared data cache access time (on-chip)	6 cycles
Memory access time	70 cycles (70 ns)
Memory interleaving	4-way
Internal Bus Parameters	
Bus Speed	1 GHz
Bus width	8 bytes
Network Parameters	
Topology	2-dimensional mesh
Flit size	8 bytes
Non-data message size	2 Flits
Router speed	250 MHz
Arbitration delay	1 router cycle
Router's internal bus width	64 bits
Channel speed	500 MHz
Channel width	32 bits

5 Simulation Environment

We have used a modified version of Rice Simulator for ILP Multiprocessors (RSIM), a detailed execution-driven simulator [34]. The modeled system is a cc-NUMA with 64 uniprocessor nodes that implements an invalidation-based, four-state MESI directory cache-coherent protocol. Table 3 summarizes the parameters of the simulated system. These values have been chosen to be similar to the parameters of current multiprocessors.

RSIM provides support for multiple memory consistency models. Although relaxed consistency models such as Processor Consistency or Release Consistency could reduce the performance impact of *Inv* and *Inv+Mem* misses, we have configured RSIM to simulate sequential consistency following the guidelines given by Hill [35].

Table 4 describes the applications we use in this study. In order to evaluate the benefits of our proposals, we have selected several scientific applications covering a variety of computation and communication patterns. BARNES-HUT, CHOLESKY, FFT, OCEAN, RADIX, WATER-SP, and WATER-NSQ are from the SPLASH-2 benchmark suite [36]. EM3D is a shared-memory implementation of the Split-C benchmark. MP3D application is drawn from the SPLASH suite [37]. Finally, UNSTRUCTURED is a computational fluid dynamics application. All experimental results reported in this paper are for the parallel phase of these applications. Data placement in our programs is either done explicitly by the programmer or by RSIM which uses a first-touch policy on a cache-line granularity. Thus, initial data-placement is quite effective in terms of reducing traffic in the system.

Through extensive simulation runs, we compare three system configurations, using the maximum number of processors available for each application (that is to say, 64 processors for all the applications except OCEAN and UNSTRUCTURED, for which a maximum of 32 processors could be simulated). The compared systems are the base

TABLE 4
Benchmarks and Input Sizes Used in This Work

Benchmark	Input Size
BARNES-HUT	8192 bodies, 4 time steps
CHOLESKY	tk15.O
EM3D	38400 nodes, degree 2, 15% remote, 25 time steps
FFT	256K complex doubles
MP3D	48000 nodes, 20 time steps
OCEAN	258x258 ocean
RADIX	2M keys, 1024 radix
UNSTRUCTURED	Mesh.2K, 5 time steps
WATER-NSQ	512 molecules, 4 time steps
WATER-SP	512 molecules, 4 time steps

TABLE 5
How *$-to-$* Misses are Satisfied

Application	LC-1 Configuration			LC-2 Configuration			UC Configuration		
	DC1	*DC2*	*MEM*	*DC1*	*DC2*	*MEM*	*DC1*	*DC2*	*MEM*
BARNES-HUT	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
CHOLESKY	63.84%	3.87%	32.29%	78.90%	7.40%	13.70%	100.00%	0.00%	0.00%
EM3D	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
FFT	11.11%	9.68%	79.21%	35.70%	46.20%	18.10%	100.00%	0.00%	0.00%
MP3D	99.72%	0.28%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
OCEAN	76.04%	2.85%	21.11%	81.67%	9.97%	8.36%	100.00%	0.00%	0.00%
RADIX	10.65%	3.03%	86.32%	17.86%	45.26%	36.88%	100.00%	0.00%	0.00%
UNSTRUCTURED	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
WATER-NSQ	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%
WATER-SP	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%	100.00%	0.00%	0.00%

TABLE 6
How *Mem* Misses are Satisfied

Application	LC-1 Configuration			LC-2 Configuration			UC Configuration		
	DC1 +SDC	*DC2 +SDC*	*MEM*	*DC1 +SDC*	*DC2 +SDC*	*MEM*	*DC1 +SDC*	*DC2 +SDC*	*MEM*
BARNES-HUT	32.22%	67.38%	0.40%	32.22%	67.38%	0.40%	32.22%	67.38%	0.40%
CHOLESKY	15.87%	49.73%	34.40%	16.29%	52.89%	30.82%	16.34%	57.96%	25.70%
EM3D	96.80%	3.20%	0.00%	96.80%	3.20%	0.00%	96.80%	3.20%	0.00%
FFT	0.04%	6.39%	93.57%	0.04%	33.20%	66.76%	0.08%	99.92%	0.00%
MP3D	16.00%	83.93%	0.07%	15.68%	84.29%	0.03%	15.68%	84.29%	0.03%
OCEAN	16.50%	31.30%	52.20%	16.53%	47.49%	35.98%	19.21%	56.43%	24.36%
RADIX	2.04%	23.75%	74.21%	2.05%	27.63%	70.32%	3.26%	33.78%	62.96%
UNSTRUCTURED	14.44%	84.26%	1.30%	14.44%	84.26%	1.30%	14.44%	84.26%	1.30%
WATER-NSQ	9.44%	90.52%	0.04%	9.44%	90.52%	0.04%	9.59%	90.37%	0.04%
WATER-SP	10.39%	89.58%	0.03%	10.39%	89.58%	0.03%	10.39%	89.58%	0.03%

system and two configurations using the node architecture presented in Section 4. The first one (*UC* system), which gives us the potential of our proposal, uses an unlimited number of entries in the first and second-level directories as well as in the shared data cache. While the second one (*LC* system) limits the number of entries in these structures. For the former three system configurations, the directory controller is assumed to be included inside the processor chip as shown in Table 3. Base system uses full-map as the sharing code for its single-level directory, which obtains the best results since unnecessary coherence messages degrading performance do not appear. As in [1], the coherence protocol used in the *UC* and *LC* configurations has been extended to support the use of a compressed third-level directory (this implies that more messages are needed to detect some race conditions) and also to include always the memory line in write-back messages and in cache-to-cache transfer responses (which increases the number of cycles needed by these messages to reach the corresponding home directory). On the contrary, the base configuration does not include these overheads.

6 Simulation Results and Analysis

In this section, we present and analyze simulation results for the base and *UC* systems as well as for two instances of the *LC* configuration. The first one, *LC-1*, limits the number of entries used in the first and second-level directories and in the shared data cache to 512, 256, and 512, respectively. This results in total sizes of less than 2 and 3 KB for the first and second-level directories, respectively, and of 32 KB for the shared data cache. The other instance, *LC-2*, increases the number of entries of all components to 1,024, resulting in total sizes of 3 KB, 10 KB, and 64 KB for the first and second-level directory and shared data cache, respectively. In all cases, the sizes of these components represent a small percentage of the L2 size (12.5 percent in the worst-case).

6.1 Impact on L2 Miss Latencies

This section analyzes how the node architecture presented in this paper impacts on the latency of each of the categories of the taxonomy presented in Section 3.

Tables 5, 6, 7, and 8 show the directory structures that are involved when solving each miss type for the *LC-1*, *LC-2*, and *UC* configurations. The first and second columns (*DC1* and *DC2*) present the percentage of misses for which directory information is obtained from the first and second-level directories, respectively (and additionally for *Mem* misses, the corresponding memory line is found in the shared data cache (*SDC*)). Thus, these misses would benefit from the novel node architecture proposed in this paper, contrary to the ones shown in the column *MEM*, for which main memory must be accessed. Note that all the accesses in the base configuration are to the main memory. As observed, for the majority of the applications, the most important fraction of the misses can be managed using information from the two first directory levels and the shared data cache, especially when the *LC-2* configuration is employed. However, some applications still require the third-level directory to provide sharing information for an important percentage of certain miss types. Note that obtaining the directory information from the third-level directory can entail two negative effects on performance: First, the miss must wait until main memory provides the corresponding directory entry and, second, in some situations, *unnecessary coherence messages* could appear as a consequence of the compressed nature of the third-level directory.

Figs. 8, 9, 10, and 11 illustrate the normalized average latency for each miss type split into network latency,

TABLE 7
How *Inv* Misses are Satisfied

	LC-1 Configuration			LC-2 Configuration			UC Configuration		
Application	*DC1*	*DC2*	*MEM*	*DC1*	*DC2*	*MEM*	*DC1*	*DC2*	*MEM*
BARNES-HUT	64.85%	35.15%	0.00%	64.85%	35.15%	0.00%	64.85%	35.15%	0.00%
CHOLESKY	86.87%	5.89%	7.24%	88.04%	7.29%	4.67%	88.37%	11.63%	0.00%
EM3D	75.43%	24.57%	0.00%	75.43%	24.57%	0.00%	75.43%	24.57%	0.00%
FFT	22.55%	12.20%	65.25%	47.47%	52.08%	0.45%	100.00%	0.00%	0.00%
MP3D	98.86%	1.14%	0.00%	98.86%	1.14%	0.00%	98.86%	1.14%	0.00%
OCEAN	84.95%	4.44%	10.61%	89.40%	7.92%	2.68%	97.38%	2.62%	0.00%
RADIX	73.14%	0.11%	26.75%	73.18%	0.09%	26.73%	92.57%	7.43%	0.00%
UNSTRUCTURED	98.52%	1.48%	0.00%	98.52%	1.48%	0.00%	98.52%	1.48%	0.00%
WATER-NSQ	96.73%	3.27%	0.00%	96.73%	3.27%	0.00%	96.73%	3.27%	0.00%
WATER-SP	51.14%	48.86%	0.00%	51.14%	48.86%	0.00%	51.14%	48.86%	0.00%

TABLE 8
How *Inv+Mem* Misses are Satisfied

	LC-1 Configuration			LC-2 Configuration			UC Configuration		
Application	*DC1*	*DC2*	*MEM*	*DC1*	*DC2*	*MEM*	*DC1*	*DC2*	*MEM*
BARNES-HUT	52.80%	47.20%	0.00%	52.80%	47.20%	0.00%	52.80%	47.20%	0.00%
CHOLESKY	95.50%	3.00%	1.50%	96.02%	3.98%	0.00%	97.13%	2.87%	0.00%
EM3D	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
FFT	14.77%	14.90%	70.33%	44.93%	54.79%	0.28%	100.00%	0.00%	0.00%
MP3D	52.29%	47.71%	0.00%	52.00%	48.00%	0.00%	52.00%	48.00%	0.00%
OCEAN	25.63%	20.50%	53.87%	39.70%	38.77%	21.53%	85.61%	14.39%	0.00%
RADIX	3.18%	0.00%	96.82%	3.18%	0.00%	96.82%	79.24%	20.76%	0.00%
UNSTRUCTURED	76.09%	23.91%	0.00%	76.09%	23.91%	0.00%	76.09%	23.91%	0.00%
WATER-NSQ	54.99%	45.01%	0.00%	54.99%	45.01%	0.00%	57.06%	42.94%	0.00%
WATER-SP	39.29%	60.71%	0.00%	39.29%	60.71%	0.00%	39.29%	60.71%	0.00%

directory latency, and miscellaneous latency (buses, cache accesses, etc.), for the base, *UC*, *LC-1*, and *LC-2* configurations. Normalized average latencies are computed dividing the average latencies for each one of the configurations by the average latencies for the base case. As can be seen from the results obtained for the *UC* configuration, the integration into the processor die of the first-level directory, and the shared data cache (*SDC*) has the potential of significantly reducing the latency of L2 misses. This is a consequence of the important reduction in the component of the latency associated with the directory.

6.1.1 *Impact on* \$-to-\$ *Miss Latencies*

As shown in Fig. 8, the small number of entries used in the *LC-1* case for the directory caches as well as for the shared data cache suffices to virtually reach the latency reductions found in the *UC* case for BARNES (66 percent), CHOLESKY (13 percent), EM3D (25 percent), MP3D (62 percent), OCEAN (40 percent), UNSTRUCTURED (17 percent), WATER-NSQ (62 percent), and WATER-SP (58 percent). As derived from Table 5, for FFT and RADIX, 80 percent or more of the misses requiring a cache-to-cache transfer need the third-level directory to provide the sharing information, which motivates the small benefit obtained by the *LC-1* configuration for these applications. The increased number of entries used in the *LC-2* case significantly reduces this percentage. Now, only 18.10 percent for FFT and 36.88 percent for RADIX of the *\$-to-\$* misses must wait for main memory latency to obtain the sharing information, and reductions of 14 percent for FFT and 11 percent for RADIX in average latency are observed. Fortunately, unnecessary coherence messages cannot appear for *\$-to-\$* misses, even when the third-level directory is accessed. This is because the compressed sharing code used by this directory level (that is to say, *BT-SuT*) is wide enough to exactly codify the identity of a single sharer, which is the case for these misses.

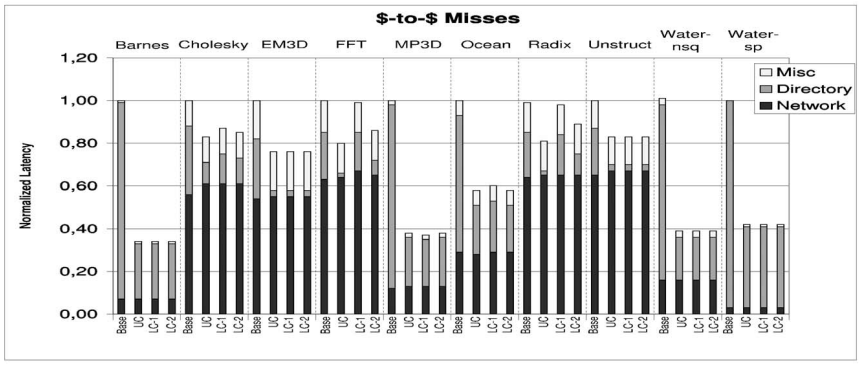

Fig. 8. Average *\$-to-\$* miss latency.

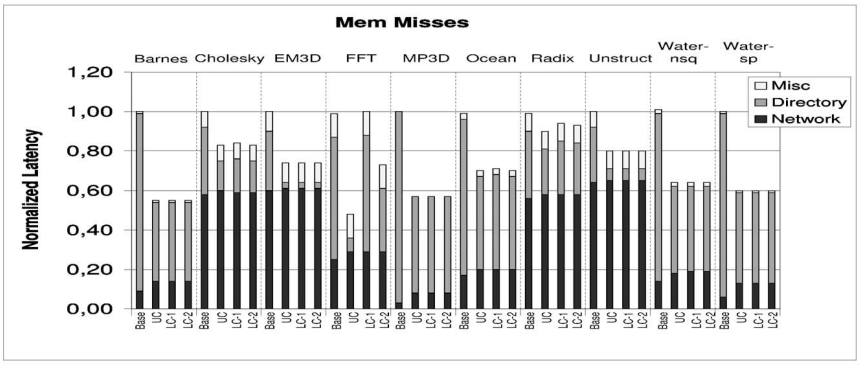

Fig. 9. Average *Mem* miss latency.

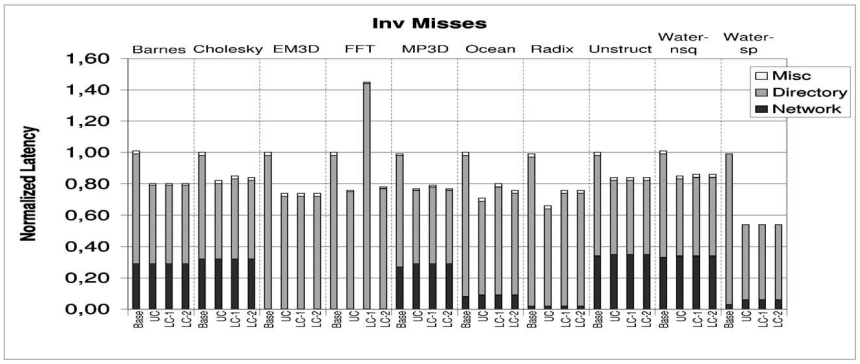

Fig. 10. Average *Inv* miss latency.

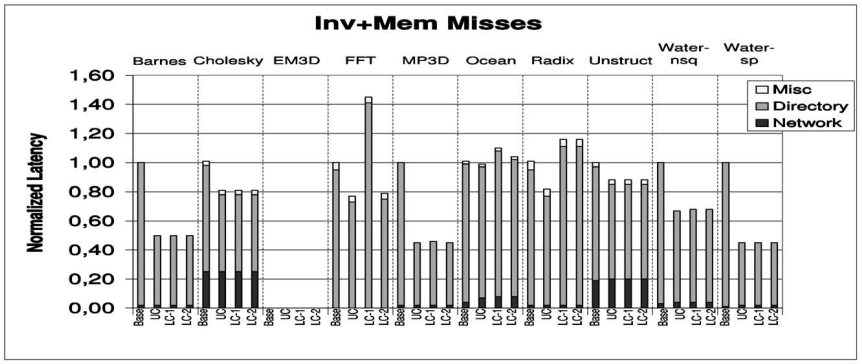

Fig. 11. Average *Inv+Mem* miss latency.

6.1.2 *Impact on* Mem *Miss Latencies*

Fig. 9 shows the normalized average latencies for *Mem* misses. For these misses, the directory must provide the memory line directly from either the shared data cache (*SDC*) or main memory. Therefore, unnecessary coherence messages cannot appear. Latency reductions of 45 percent for BARNES, 16 percent for CHOLESKY, 26 percent for EM3D, 43 percent for MP3D, 29 percent for OCEAN, 21 percent for UNSTRUCTURED, 37 percent for WATER-NSQ, and 40 percent for WATER-SP are obtained for the *LC-1* configuration. The reason for the significant reductions observed in these applications is that a large number of the *Mem* misses obtain the memory line from the shared data cache and the corresponding directory entry from one of the two first directory levels (or the own shared data cache for lines in the *Uncached* state), saving the access to the slower main memory (see Table 6). Again, the former values are very similar to those provided by the *UC* configuration. For FFT, the *LC-1* configuration is unable to reduce the latency of *Mem* misses since more than 93 percent of them must obtain the data from main memory. When moving to the *LC-2* configuration, this percentage decreases and a reduction of 28 percent is obtained. However, this reduction still remains far from the potential found for the *UC* configuration (52 percent). Finally, small latency reductions are obtained for RADIX when the *UC* configuration is used (only 10 percent). As shown in Table 6, a large percentage of the *Mem* misses suffered by this application (62.96 percent) are for lines that are accessed for the first time (cold misses), which prevents a shared data cache with an unlimited number of entries from being able to provide the memory line. The percentage of *Mem* misses satisfied by main memory is even higher for *LC-1* and *LC-2* configurations, and latency reductions of 7 percent and 8 percent, respectively, are observed.

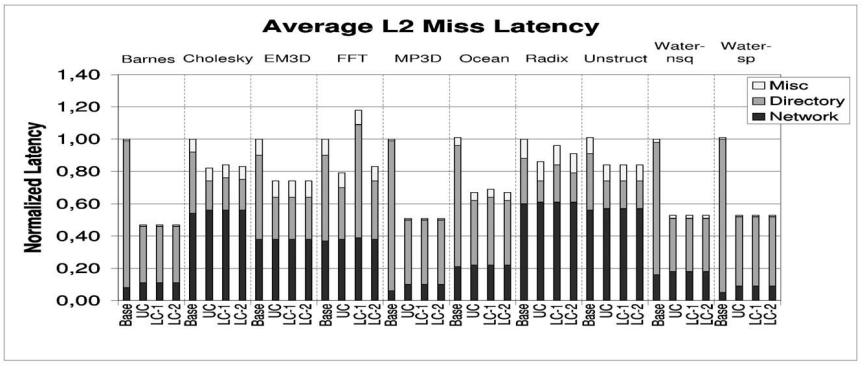

Fig. 12. Average L2 miss latency.

6.1.3 *Impact on* Inv *Miss Latencies*

Fig. 10 shows the normalized average latency for *Inv* misses. In this case, the directory must send invalidation messages and receive their corresponding acknowledgments. Therefore, the main component of the latency is caused by the directory. For *Inv* misses, *LC-1* and *UC* configurations obtain virtually identical latency reductions for BARNES (20 percent), CHOLESKY (15 percent), EM3D (26 percent), MP3D (20 percent), UNSTRUCTURED (16 percent), WATER-NSQ (15 percent), and WATER-SP (45 percent). As observed in Table 7, 26 percent of the *Inv* misses in the *LC-1* and *LC-2* configurations must obtain sharing information from main memory for RADIX application. In this case, the compressed sharing code used in the third-level directory does not increase the number of invalidation messages observed for the base case, which explains the small difference between the reduction obtained for the *UC* configuration and the ones obtained for both the *LC-1* and the *LC-2* cases (only 10 percent). For FFT, again, an important fraction of the *Inv* misses do not find directory information in the first or second-level directories when the *LC-1* configuration is used (65.25 percent). In this case, however, the compression of the third-level directory implies a loss of precision that increases the number of invalidation messages sent per invalidation event when compared with the base system and, consequently, the average time needed to satisfy these misses. This explains the significant latency increment (more than 40 percent) shown in Fig. 10. This performance degradation practically disappears when the number of entries in the first and second-level directories is increased and the *LC-2* configuration can obtain the latency reduction of 23 percent observed for the *UC* case. Finally, for OCEAN, 11 percent of the *Inv* misses obtain directory information from the third level when the *LC-1* configuration is used. Moving to *LC-2* reduces this fraction to 3 percent, and latency reductions very close to those obtained with the *UC* configuration are seen (24 percent for *LC-2* and 30 percent for *UC*).

6.1.4 *Impact on* Inv+Mem *Miss Latencies*

Fig. 11 illustrates the normalized average latency for *Inv+Mem* misses. When the *LC-1* configuration is used, important latency reductions are found for *Inv+Mem* misses for BARNES (50 percent), CHOLESKY (20 percent), MP3D (54 percent), UNSTRUCTURED (13 percent), WATER-NSQ (32 percent), and WATER-SP (55 percent), which coincide with the ones obtained for the *UC* case. As shown in Table 8, this kind of miss was not found in the EM3D application. As for the *Inv* misses, the use of the imprecise third-level directory increases the latency of *Inv+Mem* misses for some applications. In particular, when the *LC-1* configuration is used, latency degradations of more than 40 percent for FFT, 9 percent for OCEAN, and 20 percent for RADIX are observed. These degradations are completely eliminated in FFT with the use of the *LC-2* configuration and the performance of the *UC* case is obtained. However, the situation does not improve much when moving to the *LC-2* configuration in OCEAN and RADIX, and 21.53 percent and 96.82 percent, respectively, of the *Inv+Mem* misses must still obtain directory information from main memory.

6.2 Impact on Execution Time

The ability of the proposed node architecture to reduce application execution times will depend on the reductions in the average latency for each miss type, the percentage of L2 misses belonging to each category, and the weight these misses have on the application execution time.

For the applications used in our study, Fig. 13 shows the percentage of L2 misses belonging to each type, whereas Figs. 12 and 14 illustrate the normalized average L2 miss latency and execution time, respectively, for each one of the configurations considered in this paper.

As shown in Fig. 12, our proposal slightly increases the average latency due to the network in some applications (up to 4 percent for WATER-SP) when compared with the base configuration. This is caused by 1) the presence of unnecessary coherence messages and 2) the inclusion of the memory line in all the cache-to-cache transfer responses and in the write-back messages. However, important average L2 miss latency reductions are obtained for those applications that can take significant advantage of the on-chip integration of

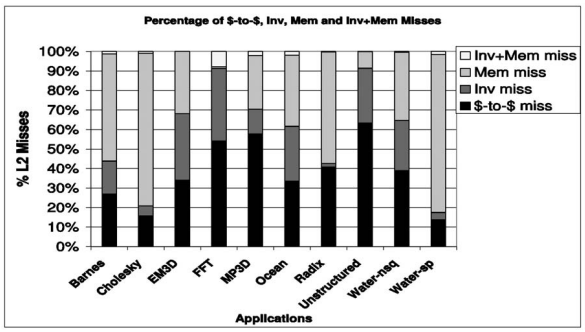

Fig. 13. Percentage of *$-to-$*, *Inv*, *Mem*, and *Inv+Mem* misses found in the application used in this paper.

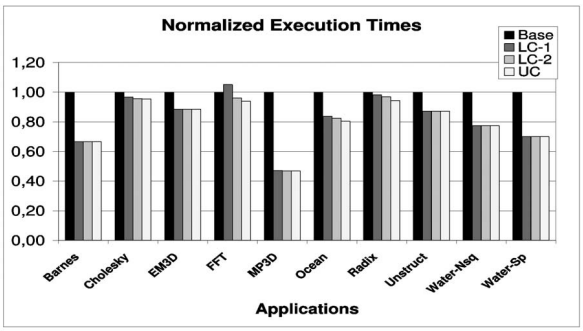

Fig. 14. Normalized execution times.

the first-level directory and the shared data cache. L2 misses have been significantly accelerated in BARNES, MP3D, OCEAN, WATER-NSQ, and WATER-SP, which motivates the important reductions on the average L2 miss latency observed for these applications (56 percent for BARNES, 49 percent for MP3D, 35 percent for OCEAN, 46 percent for WATER-NSQ, and 47 percent for WATER-SP). These reductions finally translate into significant improvements in terms of execution time (reductions of 34 percent for BARNES, 53 percent for MP3D, 20 percent for OCEAN, 22 percent for WATER-NSQ, and 30 percent for WATER-SP). More modest reductions on L2 miss latencies were found for CHOLESKY, EM3D, and UNSTRUCTURED, and reductions of 17 percent, 25 percent, and 18 percent, respectively, on average L2 miss latency are obtained, resulting in reductions of 5 percent, 11 percent, and 13 percent, respectively, on execution time. For FFT application, *Inv* and *Inv+Mem* misses represent more than 45 percent of the total miss rate, as illustrated in Fig. 13. Hence, the performance degradation observed for these misses when using the *LC-1* configuration translates into an increase on the average L2 miss latency of 18 percent and, finally, on execution time of 5 percent. This degradation disappeared when moving to the *LC-2* configuration and reductions on average L2 miss latency and execution time close to the ones reached with the *UC* configuration are obtained (execution time reductions of 4 percent and 6 percent for the *LC-2* and *UC* cases, respectively). Finally, only *Inv* misses could be significantly accelerated for RADIX application (reduction of 25 percent for the *LC-1* and *LC-2* configurations). However, as shown in Fig. 13, only 1.72 percent of the misses belong to this category, which explains the low benefit observed for the *LC-2* and *UC* configurations (reductions of 3 percent and 5 percent on execution time, respectively).

7 Conclusions

In this work, we take advantage of current technology trends and propose and study a novel node architecture especially designed to reduce the usually long L2 miss latency by significantly decreasing the component of the latency caused by the directory. Additionally, our approach minimizes the memory overhead caused by directory information.

Our proposal replaces the traditional directory with a novel three-level directory architecture and adds a small shared data to each one of the nodes that form the multiprocessor. The first-level directory as well as the small shared data cache are integrated into the processor chip of every node, which enhances performance by decreasing the time needed to obtain directory information and memory lines. On the other hand, the memory overhead entailed by directory information is significantly decreased by having two directory levels out of the processor chip.

In order to better understand the reasons for performance improvement, a taxonomy of the L2 misses according to the actions performed by the directory to satisfy them has been presented. Then, we show that a cc-NUMA multiprocessor using the proposed node architecture can achieve important reductions in the average miss latency of each one of the categories of the taxonomy when compared with a traditional cc-NUMA multiprocessor in which each node includes the coherence controller into the processor chip. Finally, these reductions translate into important improvements in the application execution times (reductions of up to 53 percent).

In general, we have found that with the use of small first and second-level directories and SDC, most L2 misses can be significantly accelerated in applications that exhibit high temporal locality on the references made by several nodes to the directory information. On the other hand, other applications such as FFT, which mainly exhibit spatial locality, require bigger structures to ensure that once a certain memory line is requested, directory information will be present when the line is subsequently referenced.

The reported improvement in performance could make our architecture competitive for medium-scale systems (64 to 256 processors) at the same time that scalability to larger systems is guaranteed. In addition, the simplicity of our proposal and the fact that it could be easily introduced in commercial processors cuts down its cost, unlike the expensive sophisticated network designs required by moderate-scale SMPs.

Acknowledgments

The authors would like to thank the anonymous referees for their detailed comments and valuable suggestions, which have helped to improve the quality of the paper. This research has been carried out using the resources of the Centre de Computació i Comunicacions de Catalunya (CESCA-CEPBA) as well as the SGI Origin 2000 of the Universitat de Valencia. This work has been supported in part by the Spanish CICYT under grant TIC2003-08154-C06-03. José Duato is supported in part by a fellowship from the Fundación Séneca (Comunidad Autónoma de Murcia, Spain).

References

[1] M.E. Acacio, J. González, J.M. García, and J. Duato, "A New Scalable Directory Architecture for Large-Scale Multiprocessors," *Proc. Seventh Int'l Symp. High Performance Computer Architecture,* pp. 97-106, Jan. 2001.

[2] M.M. Martin, D.J. Sorin, A. Ailamaki, A.R. Alameldeen, R.M. Dickson, C.J. Mauer, K.E. Moore, M. Plakal, M.D. Hill, and D.A. Wood, "Timestamp Snooping: An Approach for Extending SMPS," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 25-36, Nov. 2000.

[3] H. Hadimioglu, D. Kaeli, and F. Lombardi, "Introduction to the Special Issue on High Performance Memory Systems," *IEEE Trans. Computers,* vol. 50, no. 11, pp. 1103-1105, Nov. 2001.

[4] A. Charlesworth, "Extending the SMP Envelope," *IEEE Micro,* vol. 18, no. 1, pp. 39-49, Jan./Feb. 1998.

[5] L. Gwennap, "Alpha 21364 to Ease Memory Bottleneck," *Microprocessor Report*, vol. 12, no. 14, pp. 12-15, Oct. 1998.
[6] T. Lovett and R. Clapp, "Sting: A cc-Numa Computer System for the Commercial Marketplace," *Proc. 23rd Int'l Symp. Computer Architecture*, pp. 308-317, 1996.
[7] M.E. Acacio, J. González, J.M. García, and J. Duato, "A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors," *Proc. 16th Int'l Parallel and Distributed Processing Symp.*, Apr. 2002.
[8] The BlueGene/L Team, "An Overview of the Bluegene/L Supercomputer," *Proc. Int'l SC2002 High Performance Networking and Computing Conf.*, Nov. 2002.
[9] A. Ahmed, P. Conway, B. Hughes, and F. Weber, "AMD Opteron™ Shared Memory MP Systems," *Proc. 14th HotChips Symp.*, Aug. 2002.
[10] J. Torrellas, L. Yang, and A.T. Nguyen, "Toward a Cost-Effective DSM Organization that Exploits Processor-Memory Integration," *Proc. Sixth Int'l Symp. High Performance Computer Architecture*, pp. 15-25, Jan. 2000.
[11] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," *Proc. 27th Int'l Symp. Computer Architecture*, pp. 282-293, June 2000.
[12] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen, and K. Olukotun, "The Stanford Hydra CMP," *IEEE Micro*, vol. 20, no. 2, pp. 71-84, Mar./Apr. 2000.
[13] J. Tendler, J. Dodson, J. Fields, H. Le, and B. Sinharoy, "Power4 System Microarchitecture," *IBM J. Research and Development*, vol. 46, no. 1, pp. 5-25, Jan. 2002.
[14] P. Stenström, M. Brorsson, F. Dahlgren, H. Grahn, and M. Dubois, "Boosting the Performance of Shared Memory Multiprocessors," *Computer*, vol. 30, no. 7, pp. 63-70, July 1997.
[15] R. Iyer and L.N. Bhuyan, "Switch Cache: A Framework for Improving the Remote Memory Access Latency of cc-Numa Multiprocessors," *Proc. Fifth Int'l Symp. High Performance Computer Architecture*, pp. 152-160, Jan. 1999.
[16] R. Iyer, L.N. Bhuyan, and A. Nanda, "Using Switch Directories to Speed up Cache-to-Cache Transfers in cc-Numa Multiprocessors," *Proc. 14th Int'l Parallel and Distributed Processing Symp.*, pp. 721-728, May 2000.
[17] M.E. Acacio, J. González, J.M. García, and J. Duato, "Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in cc-Numa Multiprocessors," *Proc. Int'l SC2002 High Performance Networking and Computing Conf.*, Nov. 2002.
[18] S. Kaxiras and J.R. Goodman, "Improving cc-Numa Performance Using Instruction-Based Prediction," *Proc. Fifth Int'l Symp. High Performance Computer Architecture*, pp. 161-170, Jan. 1999.
[19] A.C. Lai and B. Falsafi, "Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction," *Proc. 27th Int'l Symp. Computer Architecture*, pp. 139-148, May 2000.
[20] M.M. Martin, P.J. Harper, D.J. Sorin, M.D. Hill, and D.A. Wood, "Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared Memory Multiprocessors," *Proc. 30th Int'l Symp. Computer Architecture*, June 2003.
[21] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M.S. Lam, "The Stanford Dash Multiprocessor," *Computer*, vol. 25, no. 3, pp. 63-79, Mar. 1992.
[22] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, M. Parkin, W. Radke, and S. Vishin, "The s3.mp Scalable Shared Memory Multiprocessor," *Proc. Int'l Conf. Parallel Processing*, pp. 1-10, July 1995.
[23] A. Gupta, W.-D. Weber, and T. Mowry, "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," *Proc. Int'l Conf. Parallel Processing*, pp. 312-321, Aug. 1990.
[24] B. O'Krafka and A. Newton, "An Empirical Evaluation of Two Memory-Efficient Directory Methods," *Proc. 17th Int'l Symp. Computer Architecture*, pp. 138-147, May 1990.
[25] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy, "The Stanford Flash Multiprocessor," *Proc. 21st Int'l Symp. Computer Architecture*, pp. 302-313, Apr. 1994.
[26] M.M. Michael and A.K. Nanda, "Design and Performance of Directory Caches for Scalable Shared Memory Multiprocessors," *Proc. Fifth Int'l Symp. High Performance Computer Architecture*, pp. 142-151, Jan. 1999.
[27] A.K. Nanda, A.-T. Nguyen, M.M. Michael, and D.J. Joseph, "High-Throughput Coherence Control and Hardware Messaging in Everest," *IBM J. Research and Development*, vol. 45, no. 2, pp. 229-244, Mar. 2001.
[28] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," *Proc. 15th Int'l Symp. Computer Architecture*, pp. 280-289, May 1988.
[29] D. Chaiken, J. Kubiatowicz, and A. Agarwal, "Limitless Directories: A Scalable Cache Coherence Scheme," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 224-234, Apr. 1991.
[30] R. Simoni and M. Horowitz, "Dynamic Pointer Allocation for Scalable Cache Coherence Directories," *Proc. Int'l Symp. Shared Memory Multiprocessing*, pp. 72-81, Apr. 1991.
[31] J. Laudon and D. Lenoski, "The SGI Origin: A ccnuma Highly Scalable Server," *Proc. 24th Int'l Symp. Computer Architecture*, pp. 241-251, June 1997.
[32] A. Gupta and W.-D. Weber, "Cache Invalidation Patterns in Shared-Memory Multiprocessors," *IEEE Trans. Computers*, vol. 41, no. 7, pp. 794-810, July 1992.
[33] D.E. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Kaufmann Publishers, Inc., 1999.
[34] V. Pai, P. Ranganathan, and S. Adve, "Rsim Reference Manual Version 1.0," Technical Report 9705, Dept. of Electrical and Computer Eng., Rice Univ., Aug. 1997.
[35] M.D. Hill, "Multiprocessors Should Support Simple Memory-Consistency Models," *Computer*, vol. 31, no. 8, pp. 28-34, Aug. 1998.
[36] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The Splash-2 Programs: Characterization and Methodological Considerations," *Proc. 22nd Int'l Symp. Computer Architecture*, pp. 24-36, June 1995.
[37] J. Singh, W.-D. Weber, and A. Gupta, "Splash: Stanford Parallel Applications for Shared-Memory," *Computer Architecture News*, vol. 20, no. 1, pp. 5-44, Mar. 1992.

Manuel E. Acacio received the MS and PhD degrees in computer science from the Universidad de Murcia, Spain, in 1998 and 2003, respectively. He joined the Computer Engineering Department, Universidad de Murcia, in 1998, where he is currently an assistant professor of computer architecture and technology. His research interests include prediction and speculation in multiprocessor memory systems, multiprocessor-on-a-chip architectures, and power-aware cache-coherence protocol design.

José González received the MS and PhD degrees from the Universitat Politecnica de Catalunya (UPC). In January 2000, he joined the Computer Engineering Department of the University of Murcia, Spain, and became an associate professor in June 2001. In March 2002, he joined the Intel Barcelona Research Center, where he is a senior researcher. Currently, he is working in new paradigms for the IA-32 family, in particular, thermal and power-aware clustered microarchitectures. He is a member of the IEEE Computer Society.

José M. García received the MS and the PhD degrees in electrical engineering from the Technical University of Valencia, in 1987 and 1991, respectively. In 1987, he joined the Computer Science Department at the University of Castilla-La Mancha at the Campus of Albacete, Spain. From 1987 to 1993, he was an assistant professor of computer architecture. In 1994, he became an associate professor at the University of Murcia, Spain. From 1995 to 1997, he served as vice-dean of the School of Computer Science. Currently, he is the director of the Computer Engineering Department, and also the head of the Research Group on Parallel Computing and Architecture. He has developed several courses on computer structure, peripheral devices, computer architecture, and multicomputer design. His current research interests include multiprocessors systems and grid computing. He has published more than 50 refereed papers in different journals and conferences in these fields. Dr. Garcia is a member of several international associations such as the IEEE and ACM, and also a member of some European associations (Euromicro and ATI).

José Duato received the MS and PhD degrees in electrical engineering from the Technical University of Valencia, Spain, in 1981 and 1985, respectively. Currently, Dr. Duato is a professor in the Department of Computer Engineering (DISCA) at the same university. He was also an adjunct professor in the Department of Computer and Information Science, The Ohio State University. His current research interests include interconnection networks, multiprocessor architectures, networks of workstations, and switch fabrics for IP routers. He has published more than 250 refereed papers. He proposed the first theory of deadlock-free adaptive routing for wormhole networks. Versions of this theory have been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E supercomputer, the internal router of the Alpha 21364 microprocessor, and the BlueGene/L supercomputer. He is the first author of the book *Interconnection Networks: An Engineering Approach*. This book was coauthored by Professor Sudhakar Yalamanchili, from the Georgia Institute of Technology and Professor Lionel Ni, from Michigan State University. Dr. Duato served as a member of the editorial boards of *IEEE Transactions on Parallel and Distributed Systems* and *IEEE Transactions on Computers*. He has been the general cochair for the 2001 International Conference on Parallel Processing and is the program committee chair for the 10th International Symposium on High Performance Computer Architecture (HPCA-10). Also, he served as cochair, member of the steering committee, vice-chair, or member of the program committee in more than 40 conferences, including the most prestigious conferences in his area (HPCA, ISCA, IPPS/SPDP, ICPP, ICDCS, Europar, HiPC). He is a member of the IEEE.