A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors

Manuel E. Acacio, José González, José M. García
Dpto. Ing. y Tecnología de Computadores
Universidad de Murcia
30071 Murcia (Spain)
{meacacio,joseg,jmgarcia}@ditec.um.es

José Duato
Dpto. Inf. de Sistemas y Computadores
Universidad Politécnica de Valencia
46071 Valencia (Spain)
jduato@gap.upv.es

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 work 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 and 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. 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 significant L2 miss latency reductions (more than 60% in some cases). These important improvements translate into reductions of more than 30% in the application execution time in some cases.

1 Introduction

Shared-memory multiprocessors cover a wide range of prices and features, from commodity SMPs to large high-performance cc-NUMA machines. The adopted architectures are quite different depending on the number of processors.

Snooping-based designs (usually known as SMPs) are the preferred architecture for machines with a small number of processors. However, some snooping-based multiprocessors have recently moved to medium scale (up to 64 processors), replacing the bus with more sophisticated interconnection network organizations. Unfortunately, the energy consumed by snoop requests, snoop bandwidth limitations, and the need to act upon all transactions at every processor, make snooping-based designs extremely challenging, especially in light of aggressive processors with multiple outstanding requests.

On the other hand, directory-based multiprocessors are much better suited for larger designs and have traditionally constituted the selected architecture for medium- and large-scale configurations. The basic idea is to keep a directory entry for every memory line, which stores the state as well as a *sharing code* indicating the caches that contain a copy of the line. L2 misses are sent to a directory controller which, in turn, using the corresponding directory entry, directs coherence transactions to the processors caching the line. However, the implementations of the shared-memory paradigm have limited scalability, then becoming infeasible for very large-scale systems, which use the message-passing paradigm. Examples of such machines are the ASCI Red, the ASCI Blue Pacific and the ASCI White multiprocessors.

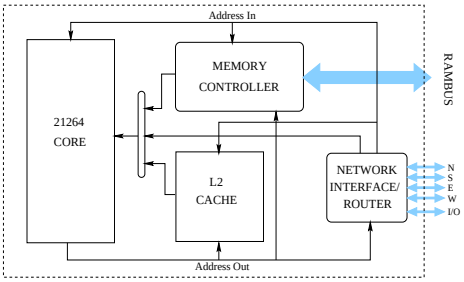

Figure 1. Alpha 21364 chip block diagram

There are two main issues limiting the scalability of cc-NUMA designs: the hardware overhead of using directories and the usually long L2 miss latencies. The most important component of the hardware overhead is the amount of memory required to store the directory information, particularly the sharing code. An effective way to reduce this overhead is by using sharing codes whose size (in bits) is not a linear function of the number of nodes, unlike the well-known *bit-vector* one [5]. Although the use of these *compressed directories* significantly reduce the memory overhead, it has also negative consequences on performance. These are caused by the appearance of *unnecessary* coherence messages, that is, coherence messages that would not be sent if a precise sharing code such as bit-vector were used. Organizing the directory as a two-level architecture was proposed in our previous work as an effective way to significantly reducing the memory overhead entailed by the directory information while achieving the performance of a non-scalable bit-vector directory [1].

Long miss latencies of directory protocols are 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 directory access. The consequences of such indirection are particularly serious for cache-to-cache transfers which, in some cases, represent more than 60% of the misses [3].

Current technology improvements allow designers to implement some key components of the system inside the processor chip. For example, the Compaq Alpha 21364 [7] includes on-chip memory controller, coherence hardware and network interface and router (see Figure 1). Taking

Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'02)
1530-2075/02 $17.00 © 2002 IEEE

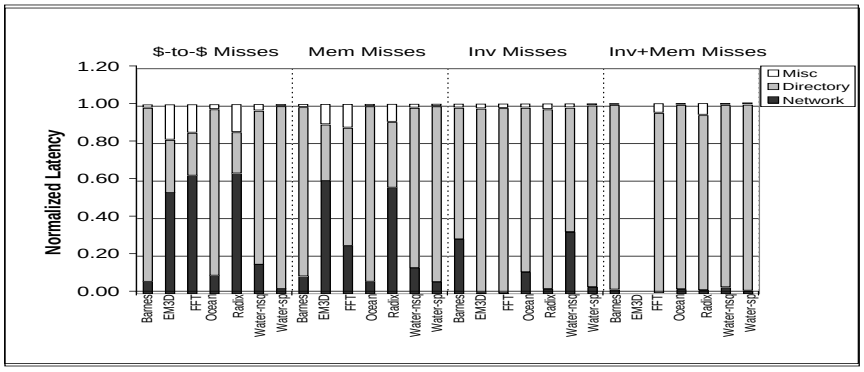

Figure 2. Normalized Average Latency for *$-to-$*, *Mem*, *Inv* and *Inv+Mem* misses

these system organizations as a starting point and considering the opportunities provided by current integration scale, in this work we propose a novel node organization especially designed to reduce the usually long L2 miss latencies and the memory overhead of using directories that characterize cc-NUMA multiprocessors and limits their scalability. Our proposal replaces the traditional directory with a novel three-level directory architecture and adds a small shared data cache to each of the nodes that form the multiprocessor. The first-level directory, which stores directory information for the most recently referenced memory lines, as well as the small shared data cache, which contains those memory lines that are expected to be requested in a near future, are integrated into the processor chip in every node, whereas second- and third-level directories are placed outside the processor. The on-chip integration of the small first-level directory and the shared data cache enhances performance whereas memory overhead is significantly reduced by having a compressed third level. 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 time needed by the home directory to satisfy them. Note that, unlike the remote data caches (RDCs) used in some systems (for example, in [11]) to cache lines that are fetched to the node from remote memories, the shared data cache proposed in our design tries to reduce the latency of accessing main memory (when needed) 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 inside the processor die.

The rest of the paper is organized as follows. Section 2 presents a classification of the L2 misses found in cc-NUMA systems. The new node architecture is proposed and justified in Section 3. Section 4 discusses our evaluation methodology. Section 5 shows a detailed performance evaluation of our novel proposal. The related work is presented in Section 6 and Section 7 concludes the paper.

2 A Taxonomy of the L2 Misses

Each time an access to a certain memory line (i.e., a load or a store) misses in the L2 cache, a request is addressed to the *home* directory of the line. The actions to be undertaken by the home directory depend on the coherence protocol, the access type and the state of the memory line.

In this section we present a taxonomy of the L2 misses found in cc-NUMA multiprocessors in terms of the actions done by the directory. Assuming that an invalidation-based four-state MESI coherence protocol is used, L2 misses can be classified into the following four categories:

1. *Cache-to-cache transfer* misses (*$-to-$*): There is a single processor caching the memory line. In this case, the directory forwards the request to the owner of the line. They are also known as *3-Hop misses*.

2. *Access to memory* misses (*Mem*): They appear for write accesses when there is not any processor caching the requested line and for read accesses when the requested line is cached by zero or more than one processor. The home directory satisfies the miss by accessing main memory in order to provide the line.

3. *Invalidate* misses (*Inv*): A write access for a memory line comes to the directory, there are several nodes holding a copy of the line and one of them is the processor issuing the access. The directory must invalidate all the copies of the memory line but the one held by the requesting processor.

4. *Invalidate and access to memory* misses (*Inv+Mem*): 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. Then, it sends a reply with the memory line to the requesting processor.

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

Table 1. Directory actions to satisfy load/store misses

Table 1 shows 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. For each one of the former categories, Figure 2 presents 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...). Further details about the evaluation methodology are included in Section 4. As can be observed, the most important fraction of the average miss latency is caused by the directory. This is true for all the applications in the *Inv* and *Inv+Mem* cases, whereas only two and three applications found network latency to exceed directory one in the *Mem* and *$-to-$* categories, respectively.

3 An Approach to Reduce L2 Miss Latencies

Previous section classified L2 misses in terms of the actions carried out by home directories to satisfy them and identified directory latency to constitute the most important component of the L2 miss latency for the majority of the applications. 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 [10]). 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 simultaneously obtained).

Assuming that nodes similar to the one presented in Figure 1 are used to form a cc-NUMA multiprocessor (as described in [7]), in this section we propose a novel node organization especially designed to reduce the latency of L2 misses by significantly decreasing the component of the latency caused by the directory. Additionally, our proposal minimizes the memory overhead caused by directory information. For a detailed description refer to [2].

3.1 Node Architecture

The proposed node organization adds several elements to the basic node architecture shown in Figure 1. In particular, the directory is organized as a *three-level* structure and a shared cache for memory lines that are frequently accessed by several nodes is included. The three-level directory architecture 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 a *N*-node system). Note that, as it is shown in [5], a small number of pointers generally suffices to keep track of the nodes caching a memory line.
2. *Second-level Directory*: It is located outside the processor chip and also has a small number of entries. In this case, a non-scalable but precise bit-vector sharing code is employed. The second-level directory can be seen as a *victim cache* of the fist level since it contains those entries that have been evicted from the first-level directory or do not fit there due to the total number of sharers is larger than the available number of pointers (three, in our case).
3. *Third-level Directory*: This level constitutes the complete directory structure (i.e., an entry per memory line) and it is located near main memory (it could be included in main memory). Each entry in this level uses a compressed sharing code to drastically reduce memory requirements. In particular, we use the Binary Tree with Subtrees (*BT-SuT*) [1] sharing code, which has space complexity O($log_2(N)$), for a *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. First-level directory has the latency of a fast on-chip cache whereas the second-level one provides data

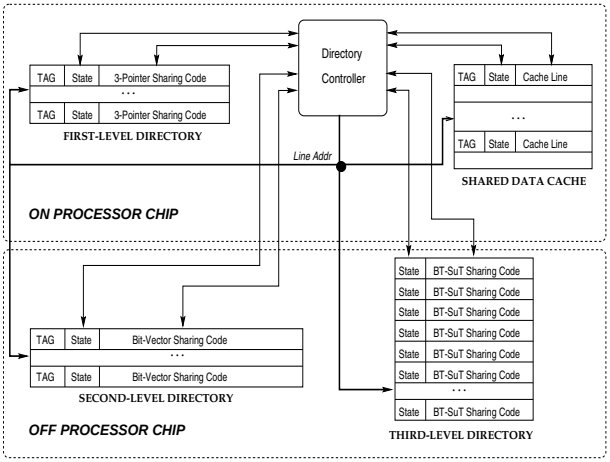

Figure 3. Proposed node architecture

at the same speed as an off-chip cache. 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, since the directory controller could quickly service them. 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.

On the contrary, *Mem* misses can not take full advantage of finding directory information in the first- or second-level directories. For this kind of misses, the directory controller must directly provide the line from main memory. In order to also accelerate *Mem* misses, our design includes a small cache inside the processor chip, the *shared data cache*. This cache stores a copy of those memory lines with a valid copy in main memory (that is, 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. State bits are also included in each entry of this shared data cache. The reason for this will be explained in next section.

Figure 3 summarizes the node architecture proposed in this work. 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 in the shared data cache in order to determine whether there is a hit.

3.2 Directory Controller Operation

Each time a L2 miss for a certain memory line arrives at the directory controller, the address of the line associated with the request is sent to each one of the directory levels as well as to the shared data cache and main memory. One of the following situations can 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. For *Mem* and *Inv+Mem* misses,

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 line in SDC (if needed)	Update entry (Sharers $>$ 3) or move entry to DC1 Insert line 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

Table 2. Directory controller operation

the access to main memory is ended if the memory line is found in the shared data cache. In all cases, the directory controller properly updates sharing information in the first- and third-level directories.

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. Now the acces to main memory is canceled when the memory line was obtained from the shared data cache or it is not needed. Sharing information in the second and third levels is also updated.
3. **The directory entry is only found in the third-level directory**. The third-level directory provides imprecise but correct sharing information.

On a 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. 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.
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]. 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. This way an entry in the first- or second-level directories will not be wasted.
3. On the response to a *$-to-$* miss indicating that a cache-to-cache transfer was performed, when the miss was caused by a load instruction. The state of the line was changed from *Private* to *Shared*, and the acknowledgment message contains a copy of the line[1].

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. An entry in the shared data cache is freed in two cases: first, when a processor obtains an exclusive copy of the memory line, 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 ensured to be 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 entries associated to memory lines with pending coherence transactions. Table 2 summarizes the most important issues described in this section.

3.3 Implementation Issues

In this work we assume the organization of the first- and second-level directory caches as well as of the shared data cache to be fully associative, with a LRU replacement policy[2]. Each line in the first- and second-level directories contains a single directory entry. 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 the directory controller to provide 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 [4], it could be directly stored in main memory by computing ECC at a coarser granularity and utilizing the unused bits.

4 Simulation Environment

We have used a modified version of Rice Simulator for ILP Multiprocessors (RSIM), a detailed execution-driven simulator [14]. The modeled system is a cc-NUMA with 64 uniprocessor nodes that implements an invalidation-based, four-state MESI directory-based cache coherence protocol and sequential consistency. Table 3 summarizes the parameters of the simulated system. These values have been chosen to be similar to the parameters of current multiprocessors.

In order to evaluate the benefits of our proposals, we have selected several scientific applications covering a variety of computation and communication patterns. *Barnes* (8,192 Bodies,4 timesteps), *FFT* (256K complex doubles), *Ocean* (258$\times$258 grid), *Radix* (2M keys,1024 radix), *Water-Spatial* (512 molecules,4 timesteps) and *Water-Nsquared* (512 molecules,4 timesteps) are from the SPLASH-2 benchmark suite [16]. *EM3D* (38,400 nodes,

[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.

[2]Practical implementations can be set-associative, achieving similar performance at lower cost [12].

64-Node System	
ILP Processor Speed	1 GHz
Cache line size	64 bytes
L1 cache WT	Direct mapped, 32KB
L1 hit time	2 cycles
L2 cache WB	4-way associative, 512KB
L2 hit time	15 cycles, pipelined
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
Shared data cache access time (on-chip)	6 cycles
Memory access time	70 cycles (70 ns)
Bus speed/width	1 GHz/8 bytes
Network Topology	2-dimensional mesh
Flit size	8 bytes
Non-data message size	2 Flits
Router speed	250 MHz
Channel speed	500 MHz
Channel width	32 bits

Table 3. Base system parameters

25 time steps) is a shared-memory implementation of the Split-C benchmark. All experimental results reported in this paper are for the parallel phase of these applications.

Using these applications we compare, through extensive simulation runs, three configurations: the *Base* system (directory information at main memory) and two configurations using the node architecture presented in Section 3. 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. The second one (*LC* system) limits the number of entries in these structures. The directory controller is assumed to be included inside the processor chip as shown in Table 3. *Base* system uses bit-vector 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 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.

5 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 for all the components to 1024, resulting in total sizes of 3 KB, 10 KB and 64 KB for the first-level directory, second-level directory and shared data cache, respectively. In all cases, the sizes of these components represent a small percentage of the L2 size (a 12.5% in the worst case).

5.1 Impact on L2 Miss Latencies

This section analyzes how the node architecture presented in this work impacts on the latency of L2 misses.

Tables 4 to 7 show the directory structures involved when satisfying 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, for *Mem* misses, the cache line from the shared data cache (*SDC*)). Thus, these misses would benefit from the novel node architecture proposed in this work, 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.

Figures 4 to 7 illustrate the normalized average latency for each miss type split into network latency, directory latency and miscellaneous latency (buses, cache accesses...), 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.

Impact on *$-to-$* Miss Latencies

As shown in Figure 4, 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%), EM3D (25%), Ocean (42%), Water-nsq (62%) and Water-sp (58%). As derived from Table 4, for FFT and Radix, more than 80% of the misses requiring a cache-to-cache transfer need the third-level directory to provide the sharing information. The increased number of entries used in the *LC-2* case significantly reduces this percentage. Now, only 18.10% for FFT and 36.88% for Radix of the *$-to-$* misses must wait during main memory latency to obtain the sharing information and reductions of 14% for FFT and 11% for Radix in average latency are observed. 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. This second effect can not actually take place for *$-to-$* misses since the compressed sharing code used by the third-level directory is wide enough to exactly codify the identity of a single sharer.

Impact on *Mem* Miss Latencies

Figure 5 shows the normalized average latencies for *Mem* misses. For this kind of misses unnecessary coherence messages can not appear. Latency reductions of 45% for Barnes, 26% for EM3D, 32% for Ocean, 37% for Water-nsq and 40% for Water-sp are obtained for the *LC-1* configuration. The reason for these significant reductions is that a large number of the *Mem* misses found the memory line in the shared data cache and the sharing information in one of the two first directory levels, saving the access to the slower main memory (see Table 5). Again, the former values are very similar to those reached with the *UC* configuration. For FFT, the *LC-1* configuration is unable to reduce the latency of *Mem* misses since more than 93% of them must obtain the data from main memory. When moving to the *LC-2* configuration, this percentage decreases and a reduction of 28% is obtained. However, this reduction still remains far from the potential found for the *UC* configuration (52%). Finally, small latency reductions are obtained for Radix when the *UC* configuration is used (only a 10%). As shown in Table 5, a large percentage of the *Mem* misses suffered by this application (62.96%) are for lines that are accessed for the first time, which prevents a shared data cache with an unlimited number of entries from being able to provide the memory line. The percentage of *Mem*

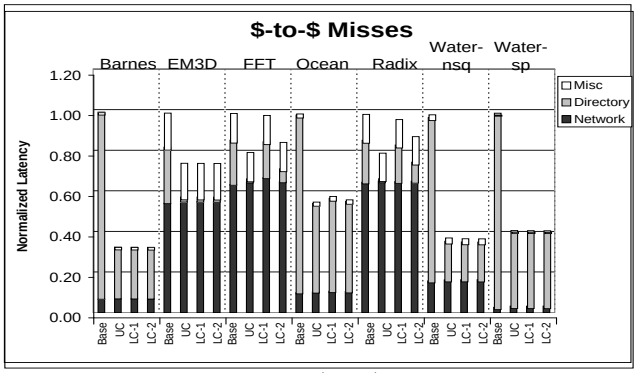

Figure 4. Average *$-to-$* miss latency

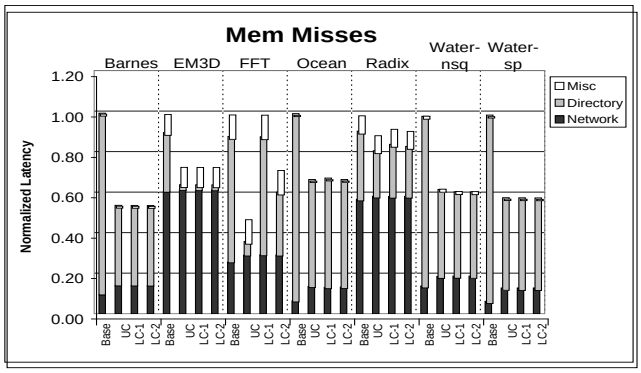

Figure 5. Average *Mem* miss latency

	LC-1 Configuration			LC-2 Configuration			UC Configuration		
Application	*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%
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%
Ocean	81.66%	5.18%	13.16%	95.60%	3.15%	1.25%	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%
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 4. How *$-to-$* misses are satisfied

	LC-1 Configuration			LC-2 Configuration			UC Configuration		
Application	*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%
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%
Ocean	32.35%	48.26%	19.39%	32.66%	48.51%	18.83%	34.06%	50.43%	15.51%
Radix	2.04%	23.75%	74.21%	2.05%	27.63%	70.32%	3.26%	33.78%	62.96%
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%

Table 5. How *Mem* 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%
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%
Ocean	83.12%	9.38%	7.50%	91.70%	6.94%	1.36%	93.79%	6.21%	0.00%
Radix	73.14%	0.11%	26.75%	73.18%	0.09%	26.73%	92.57%	7.43%	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 6. 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	52.80%	47.20%	0.00%	52.80%	47.20%	0.00%	52.80%	47.20%	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%
Ocean	56.69%	39.97%	3.34%	59.27%	40.00%	0.73%	60.49%	39.51%	0.00%
Radix	3.18%	0.00%	96.82%	3.18%	0.00%	96.82%	79.24%	20.76%	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%

Table 7. How *Inv+Mem* misses are satisfied

misses satisfied by main memory is even higher for *LC-1* and *LC-2* configurations, and latency reductions of 7% and 8%, respectively, are observed.

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

Figure 6 shows the normalized average latency for *Inv* misses. In this case, the directory must send invalidation messages and receive their corresponding *acks*. 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%), EM3D (26%), Water-nsq (15%) and Water-sp (45%). As observed in Table 6, 26% of the *Inv* misses in the *LC-1* and

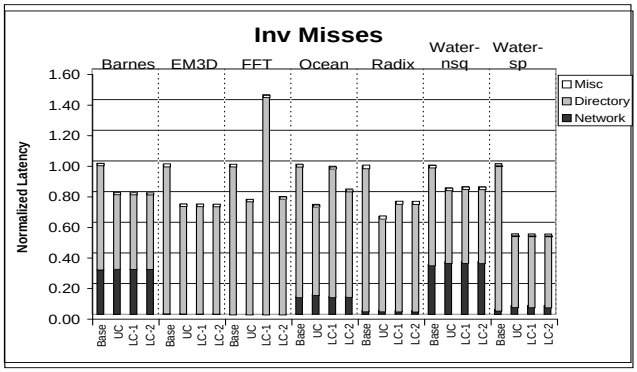

Figure 6. Average *Inv* miss latency

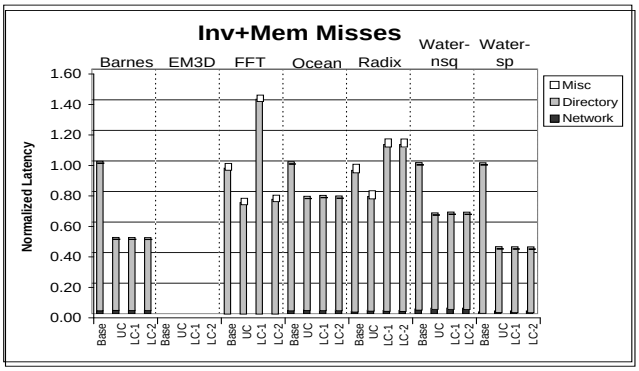

Figure 7. Average *Inv+Mem* miss latency

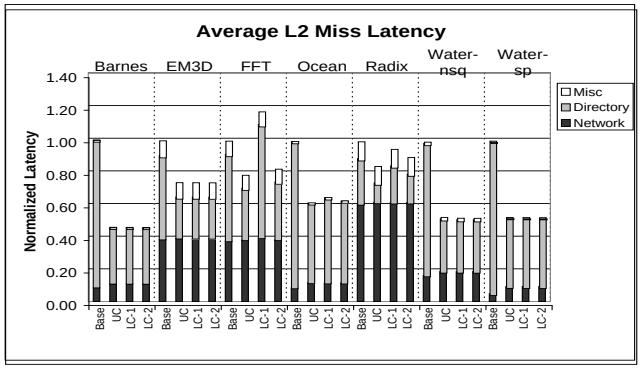

Figure 8. Average L2 miss latency

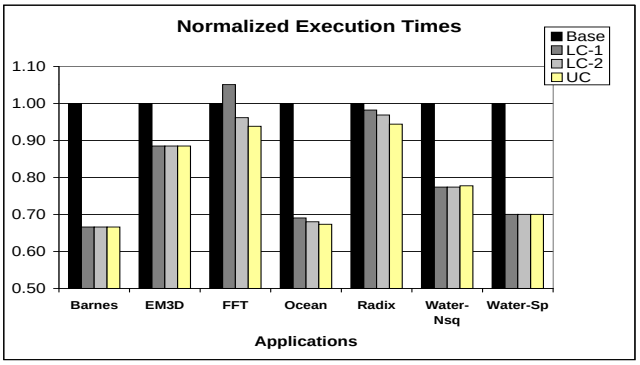

Figure 9. Normalized execution time

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%). For FFT, again, an important fraction of the *Inv* misses found the directory information in main memory when the *LC-1* configuration was used (65.25%). However, in this case, the compression of the third-level directory implies a loss of precision that increases the number of invalidation messages for each invalidation event when comparing with the *Base* system and, consequently, the average time needed to satisfy this kind of misses. This explains the significant latency increment (more than 40%) shown in Figure 6. This 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% observed for the *UC* case. A similar situation also appears for Ocean. In this case, the latency reduction obtained with the *UC* configuration (26%) can not be reached when using the *LC-2* one (17%).

Figure 7 illustrates the normalized average latency for *Inv+Mem* misses. When using the *LC-1* configuration, important latency reductions are also found for *Inv+Mem* misses for Barnes (50%), Ocean (22%), Water-nsq (32%) and Water-sp (55%), which coincide with the ones obtained for the *UC* case. As shown in Table 7, this kind of miss was not found in EM3D application. As in the *Inv* misses case, the use of the imprecise third-level directory increases the latency of *Inv+Mem* misses for FFT (more than 40%) and Radix (20%) when the *LC-1* configuration is used. This degradation is completely eliminated in FFT with the use of the *LC-2* configuration and the performance of the *UC* case is obtained. However, when moving to the *LC-2* configuration in Radix the situation does not improve and a 96.82% of the *Inv+Mem* misses must still obtain directory information from main memory.

5.2 Impact on Execution Time

For the applications used in our study, Table 8 shows the percentage of L2 misses belonging to each type, whereas Figures 8 and 9 present the normalized average L2 miss latency and execution time, respectively, for each one of the configurations considered in this work.

Application	$-to-$	Mem	Inv	Inv+Mem
Barnes-Hut	26.43%	55.04%	17.02%	1.51%
EM3D	34.07%	31.86%	34.07%	0.00%
FFT	54.12%	0.74%	37.22%	7.92%
Ocean	42.54%	23.39%	33.65%	0.53%
Radix	40.81%	57.14%	1.72%	0.33%
Water-nsq	39.08%	34.62%	25.74%	0.57%
Water-sp	12.45%	82.33%	3.64%	1.58%

Table 8. L2 misses according to directory actions

Important average L2 miss latency reductions are obtained for those applications that can take significant advantage of the on-chip integration of the first-level directory and the shared date cache. L2 misses have been significantly accelerated in Barnes, Ocean, Water-nsq and Water-sp, which motivates the important reductions on the average L2 miss latency observed for these applications (56% for Barnes, 36% for Ocean, 46% for Water-nsq and 47% for Water-sp). These reductions finally translate into significant improvements in terms of execution time (reductions of 34% for Barnes, 31% for Ocean, 22% for Water-nsq and 30% for Water-sp). More modest reductions on L2 miss latencies were found for EM3D and a reduction of 25% on average L2 miss latency is obtained, resulting in a reduction of 11% on execution time. For FFT application, the performance degradation observed for *Inv* and *Inv+Mem* misses

when using the *LC-1* configuration disappears when moving to the *LC-2* configuration and reductions on average L2 miss latency and execution time close to those reached with the *UC* configuration are obtained. Finally, only *Inv* misses could be significantly accelerated for Radix application. However, as shown in Table 8, only 1.72% of the misses belong to this category.

6 Related Work

In order to significantly reduce the memory overhead entailed by the directory while achieving the same performance as a non-scalable bit-vector directory, we proposed in [1] a *two-level directory* architecture, which combined a small first-level directory (a few bit-vector entries for the most recently accessed lines) with a compressed second-level directory, with one entry per memory line.

Some previously proposed designs, such as the Compaq Alpha 21364 [7] or the Compaq Piranha CMP [4], already include the coherence hardware inside the processor die. However, in such designs, directory information is stored in main memory, which puts main memory latency into the critical path of L2 misses.

Caching directory information was originally proposed in [6] and [13] as a means to reduce the memory overhead entailed by directories. More recently, directory caches have also been used to reduce directory access times [9][12]. In addition, remote data caches (RDCs) have also been used in several designs (as [11]) to accelerate the access to remote data. A RDC caches remote data in local memory resources acting as backup for the processor caches. Multiprocessor-on-a-chip is emerging as an active research topic nowadays. Examples of such architectures can be found in [4][8]. Finally, Torrellas *et al.* [15] explore how a cache-coherent DSM machine built around Processor-In-Memory chips might be cost-effectively organized.

7 Conclusions

We take advantage of current technology trends proposing and studying in this work a novel node architecture especially designed to reduce the 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 cache 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, whereas the second- and third-level directories are placed outside the processor. The on-chip integration of the small first-level directory (which uses a limited number of pointers as sharing code) and the shared data cache ensures performance. The third-level directory is a complete directory structure (one entry per memory line) that uses our *BT-SuT* compressed sharing code to drastically reduce memory requirements and the second-level directory is a small directory cache (using bit-vector sharing code) that tries to minimize the negative effects of having an imprecise third level as well as to quickly provide directory information when it is not present in the first level.

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. Latency reductions up to 66% for *$-to-$* misses, 45% for *Mem* misses, 45% for *Inv* misses and 55% for *Inv+Mem* misses have been obtained. These reductions translate into important improvements in the application execution times (reductions up to 34%).

Acknowledgments

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 (grant TIC2000-1151-C07).

References

[1] M. E. Acacio, J. González, J. M. García and J. Duato. "A New Scalable Directory Architecture for Large-Scale Multiprocessors". *7th Int'l Symposium on High Performance Computer Architecture*, Jan. 2001.

[2] M. E. Acacio, J. González, J. M. García and J. Duato. "A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors". *Tech. Report UM-DITEC-2002-1 , Computer Engineering Department, University of Murcia*, Jan. 2002.

[3] L. A. Barroso, K. Gharachorloo and E. Bugnion. "Memory System Characterization of Commercial Workloads". *25th Int'l Symposium on Computer Architecture*, June 1998.

[4] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzky, S. Qadeer, B. Sano, S. Smith, R. Stets and B. Verghese. "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing". *27th Int'l Symposium on Computer Architecture*, June 2000.

[5] D. E. Culler, J. P. Singh and A. Gupta. *"Parallel Computer Architecture: A Hardware/Software Approach"*. Morgan Kaufmann Publishers, Inc., 1999.

[6] A. Gupta, W.-D. Weber and T. Mowry. "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes". *Int'l Conference on Parallel Processing*, August 1990.

[7] L. Gwennap. "Alpha 21364 to Ease Memory Bottleneck". *Microprocessor Report*, pp. 12–15, October 1998.

[8] L. Hammond, M. Willey and K. Olukotun. "The Standford Hydra CMP". *Proc. of Hot Chips 11*, August 1999.

[9] 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". *21st Int'l Symposium on Computer Architecture*, Apr. 1994.

[10] J. Laudon and D. Lenoski. "The SGI Origin: A ccNUMA Highly Scalable Server". *24th Int'l Symposium on Computer Architecture*, June 1997.

[11] T. Lovett and R. Clapp. "STiNG: A CC-NUMA Computer System for the Commercial Marketplace". *23rd Int'l Symposium on Computer Architecture*, 1997.

[12] M. M. Michael and A. K. Nanda. "Design and Performance of Directory Caches for Scalable Shared Memory Multiprocessors". *5th Int'l Symposium on High Performance Computer Architecture*, Jan. 1999.

[13] B. O'Krafka and A. Newton. "An Empirical Evaluation of Two Memory-Efficient Directory Methods". *17th Int'l Symposium on Computer Architecture*, May 1990.

[14] V. Pai, P. Ranganathan and S. Adve. "RSIM Reference Manual version 1.0". *Tech. Report 9705, Department of Electrical and Computer Engineering, Rice University*, Aug. 1997.

[15] J. Torrellas, L. Yang and A. T. Nguyen. "Toward A Cost-Effective DSM Organization That Exploits Processor-Memory Integration". *6th Int'l Symposium on High Performance Computer Architecture*, Jan. 2000.

[16] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". *22nd Int'l Symposium on Computer Architecture*, June 1995.