Distributed Cooperative Caching

Enric Herrero
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
eherrero@ac.upc.edu

José González
Intel Barcelona Research Center
Intel Labs-Universitat Politècnica de Catalunya
pepe.gonzalez@intel.com

Ramon Canal
Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
rcanal@ac.upc.edu

ABSTRACT

This paper presents the Distributed Cooperative Caching, a scalable and energy-efficient scheme to manage chip multiprocessor (CMP) cache resources. The proposed configuration is based in the Cooperative Caching framework [3] but it is intended for large scale CMPs. Both centralized and distributed configurations have the advantage of combining the benefits of private and shared caches. In our proposal, the Coherence Engine has been redesigned to allow its partitioning and thus, eliminate the size constraints imposed by the duplication of all tags. At the same time, a global replacement mechanism has been added to improve the usage of cache space. Our framework uses several Distributed Coherence Engines spread across all the nodes to improve scalability. The distribution permits a better balance of the network traffic over the entire chip avoiding bottlenecks and increasing performance for a 32-core CMP by 21% over a traditional shared memory configuration and by 57% over the Cooperative Caching scheme.

Furthermore, we have reduced the power consumption of the entire system by using a different tag allocation method and by reducing the number of tags compared on each request. For a 32-core CMP the Distributed Cooperative Caching framework provides an average improvement of the power/performance relation ($MIPS^3/W$) of 3.66x over a traditional shared memory configuration and 4.30x over Cooperative Caching.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C.1.4 [**Processor Architectures**]: Parallel Architectures—*Distributed architectures*

General Terms

Design, Management, Performance

PACT'08, October 25–29, 2008, Toronto, Ontario, Canada.

Keywords

Chip multiprocessors, memory hierarchy, distributed cooperative caching, energy efficiency

1. INTRODUCTION

Over the last few years, several chip multiprocessors have appeared in the market. These configurations make an efficient use of the growing silicon real estate and try to take advantage of the existing parallelism of applications. Server and high-end applications are the most benefited from these platforms and it is also expected that future desktop applications for recognition, mining and synthesis [8] are going to require a high number of cores. Existing roadmaps and research trends like Intels Tera-scale [22] processor show that the number of cores is going to increase in the future. These architectures are going to exacerbate existing challenges such as power dissipation, wire delays and off-chip memory bandwidth. In this environment, a power and performance optimized memory hierarchy is crucial. Such configuration must minimize off-chip misses by optimizing on-chip memory usage and also reduce miss latency by placing data close to the requester.

Cache coherence in multiprocessors has been traditionally granted either by snoop or directory based schemes. Directory based schemes need an extra level of indirection but have a better scalability and can provide the best configuration for multiprocessors with a high number of cores. When designing the on-die L2 cache, two different alternatives come up, private and shared caches.

Logically-unified shared L2 cache provides a good response for processors with a reduced number of cores because the global number of L2 misses is usually smaller and they make an efficient use of the available L2 cache space. This is the most common organization for the last-level cache due to its simplicity and good performance [7, 19]. However, for a higher number of cores this type of configuration generates a bottleneck in centralized aggregate caches or produces a high demand on the interconnection network for distributed Non Uniform Cache Access (NUCA) architectures. The network usage increase has three negative effects; it increases the overall power consumption, requires a network with higher bandwidth and increases the miss latency.

Private L2 caches, on the other hand, provide a uniform and usually lower latency since data is stored in the local nodes. These configurations have the additional advantage of avoiding inter-core cache conflicts. However, since not all threads running in the cores have the same cache requirements, this causes an inefficient use of L2 cache space, and

these caches often require a higher number of off-chip accesses with the inherent latency penalizations.

To find a compromise between these two solutions several proposals have appeared that try to achieve the latency of private configurations and the low number of off-chip accesses of shared configurations [5, 3, 10, 18, 24]. One of the most interesting is the Cooperative Caching framework [3]. This organization duplicates all cache tags in a centralized coherence engine to allow block sharing between nodes and reduce off-chip misses. Furthermore, it uses private L2 caches to allocate blocks closer to the requester and reduce the L1 miss latency. This technique, however, does not allow an efficient use of all the cache space so it also implements a forwarding mechanism for the evicted blocks. This mechanism spills replaced blocks to other L2 caches to avoid future off-chip accesses.

This organization has two main limitations. First, the centralized structure of the replicated tags becomes a bottleneck for a high number of nodes; and second, the coherence engine -even if banked- may have a particular address in any of them. This means that all banks must be accessed on each request and a high number of tags must be compared, increasing the power consumption significantly for a system with many processors.

We propose the Distributed Cooperative Caching scheme in order to overcome the power and scalability issues of the Cooperative Caching framework. We have redesigned the coherence engine structure and its allocation mechanism to allow a distribution of the replicated tags across the chip. Our allocation mechanism reduces the number of tags checked on every request thus reducing the energy consumption. Another benefit of our organization is the possibility of having a smaller number of replicated tags while the centralized Cooperative Caching requires a replica for every cache tag. The tag entry of a block that is shared by several caches in the Cooperative Caching framework is going to be allocated in all the tag replicas while in our proposal only one entry is going to be used for each block, making a more efficient use of space. We will show that our organization gets an average performance improvement over the Cooperative Caching of 57% and a MIPS^3/W relation improvement of 4.30x for a 32-core CMP thanks to a request distribution and bottleneck avoidance.

This paper makes the following contributions:

- We present a comparison of the power/performance tradeoffs for different configurations of the memory hierarchy of CMPs with a high number of cores. We show that configurations with shared last-level caches have very high bandwidth requirements for big multiprocessors. This results in a performance degradation and a tremendous increase of the energy consumption. We also show that private configurations are the least energy consuming but have significant performance penalties for a large number of cores due to the inability to make an efficient use of the last level cache.

- We propose the Distributed Cooperative Caching framework, a new scalable and power efficient configuration for chip multiprocessors. We present a new Coherence Engine design that reduces power consumption and provides better scalability. This configuration makes possible an effective sharing of all the cache entries while keeping blocks close to the requesting nodes.

2. DISTRIBUTED COOPERATIVE CACHING

2.1 Cooperative Caching

Our proposal is based on the Cooperative Caching (CC) framework [3] proposed by Chang and Sohi. CMP Cooperative Caching tries to create a globally-managed, "shared", aggregate on-chip cache with private caches. The main objectives of this configuration are to reduce the average miss latency by approaching the blocks to the local node, to improve the cache utilization and to achieve as much performance isolation between nodes as possible. The hardware requirements of this approach are a central directory with a duplicate of all the L1 and L2 cache tags. This directory (Central Coherence Engine) is the responsible for maintaining blocks coherent and to share the blocks between caches. Figure 1 shows the memory configuration of this organization.

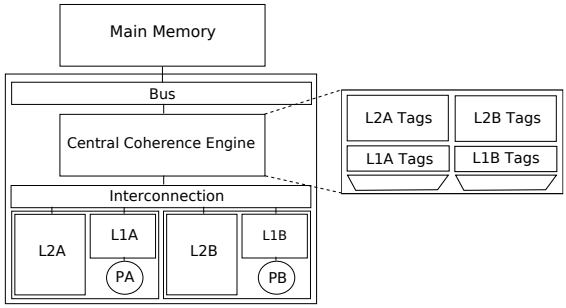

Figure 1: CC Memory Structure.

The working principle of this approach is that all the L2 misses are handled by the CCE -which keeps a copy of all the tags. If a particular cache access misses in the local L2 and the block is stored in another cache, the CCE is going to have a hit and the request will be forwarded to the owner. Then the data is sent through a cache-to-cache transfer and the CCE is acknowledged of the end of the transaction.

To be able to use efficiently the cache space, the cooperative caching also implements the N-Chance Forwarding algorithm for replacements. When a block is evicted from an L2 the CCE forwards it to another cache if it is the last copy in the chip. To avoid infinite forwardings a counter is set for each block. By default each block is forwarded N times before being evicted from the chip and if the block is reused the counter is reset. To avoid a chain reaction of replacements a spilled block is not allowed to trigger a subsequent spill. When applied to CMP Cooperative Caching N is set to 1 since a replication control is already employed and further spilling would degrade performance by evicting newer blocks.

This approach, however, has some limitations that we try to overcome with our proposal. The first one is that a centralized directory presents important restrictions to the scalability of the multiprocessor. The centralized nature of the coherence engine limits the number of processors that can handle without creating a bottleneck and degrading the performance. The second limitation of this configuration is the

power consumption of the centralized directory. The number of tags that must be checked on each request increases with the number of nodes, raising also the overall power consumption. Making the CCE scalable is somewhat challenging since the centralized version is already banked and does not behave well for a high number of processors. In the next section we will further discuss these limitations and our suggested solutions.

2.2 The Distributed Cooperative Caching scheme

We propose the Distributed Cooperative Caching (DCC) scheme in order to solve the scaling issues of the previous configuration. We have redesigned the coherence engine to be able to distribute it across all the nodes. This avoids bottlenecks and limits the number of tag checks that must be done on each request.

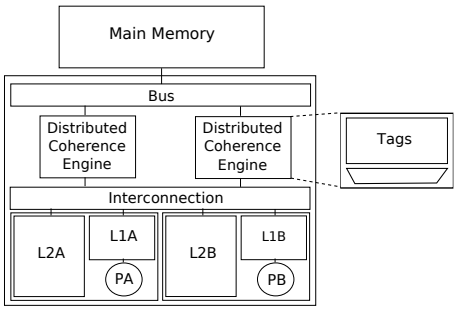

Figure 2: DCC Memory Structure.

In our approach, the Coherence Engine is partitioned into several Distributed Coherence Engines (DCE) that are responsible for a portion of the address space. The number of DCEs and the number of entries each one has does not depend on the cache sizes. Figure 2 shows the memory structure used in our proposal. Addresses in the Coherence Engines are mapped in an interleaved way. On a local L2 cache miss, the corresponding DCE for that address is accessed and if the cache entry is found the request is redirected to the owner.

The organization of the directory in the Distributed Coherence Engine is completely different to the one in the Cooperative Caching scheme. In the DCC framework, tags are distributed in an interleaved way across the DCEs in order to distribute DCC requests across the network and hence avoid bottlenecks. This distribution implies that, unlike the centralized configuration that has a tag for each cache entry, tag entries are allocated just in one DCE depending on its address. As a result, if the entries are not perfectly distributed in the address space, we can have more entries in the caches than in the DCE and a DCE replacement is going to be triggered. Because of that, it is necessary to extend the coherence protocol to be able to handle the invalidation of cache blocks due to DCE replacements.

Figure 3 shows the organization of the Coherence Engine for both Centralized and Distributed versions of Cooperative Caching. We can see that the organization of the centralized version is formed by a unique structure that has the replicated tags distributed in banks, each one representing a cache. In the DCC, we have an arbitrary number of Coherence Engines that store tags from all caches. We can also

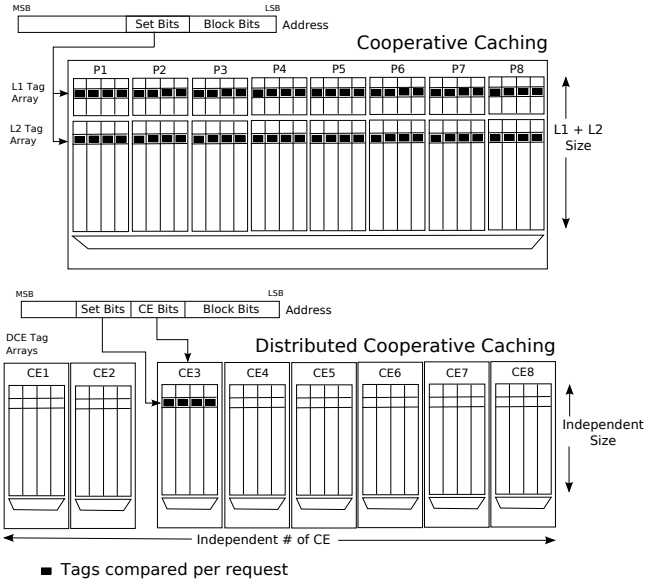

Figure 3: Directory structures.

see in the figure that the number of tags compared for every request is significantly smaller in the DCC scheme, and this results in a reduced energy consumption. In the Distributed version the number of checked tags depends on the associativity that we want to assign to the DCEs. On the other hand, for the Centralized version, the number of tags compared depends on the number of processors and the associativity of their caches. This number increases with the number of nodes; making this configuration suitable only for a CMP with a reduced number of processors. The example of Figure 3 shows the Coherence Engines of an 8-core CMP with 4-way associative caches. We can observe that for the centralized version 64 tags are compared while for the distributed version only 4 are compared.

In addition to all these benefits, the Distributed Cooperative Caching also allows hardware design reuse since its modular and scalable structure can be replicated as we add more processors on a chip.

2.2.1 *The DCE replacement policy: an example*

To show the benefits of the DCE tag replacement policy, Figure 4 demonstrates the working principle of the Centralized and the Distributed versions of Cooperative Caching.

The situation depicted shows the L2 caches and Coherence Engines of a system with two nodes (A and B) for simplicity. It considers the situation of two threads, one per node, that make an extensive use of their caches. It is also considered that node A always makes requests slightly before than node B. Blocks in the cache are represented by the letter of the requesting node and a number that indicates the time when that block was requested. We start in a warmed-up situation where both caches are full to see how replacements are handled.

In the upper part of the figure the behaviour of the Centralized Cooperative Caching is shown. Let's suppose that node A makes a request for a new block (Action 1). In this case, since the block is not in the local L2, the CCE is checked. Since the block is neither in any other cache, memory is accessed. Block A5 is then sent to the requester

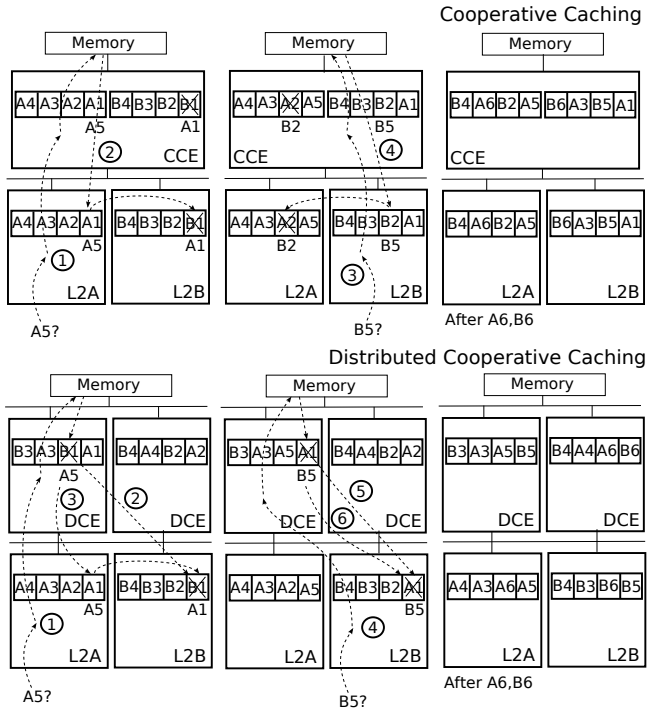

Figure 4: Working Example.

(Action 2). Since there is not enough space, block A1 is spilled to node B. Block B1 is evicted from the chip since subsequent spillings are not allowed.

In the second request, node B asks also for a block to the CCE (Action 3). Request is forwarded to memory that sends the block to the requester (Action 4). Since there is not enough place, a replacement is done. The locally oldest block, B2, is spilled to node A; evicting from the chip A2.

In the bottom part of Figure 4 the behaviour of the Distributed Cooperative Caching is depicted. As in the previous case block A5 is requested (Action 1), but now to the corresponding DCE. Since the block is not in any other cache, memory is accessed. In this configuration when the block is sent to the DCE it generates an eviction. In order to make the example more interesting, although the result is the same, block B1 is replaced, invalidating the entry in the L2 (Action 2). Then the block is allocated in the corresponding DCE and sent to the requesting node (Action 3). Since the cache is full, block A1 is spilled to node B and is placed in the invalidated entry.

In the second request node B accesses also the DCE and memory asking for the block (Action 4). When block B5 is allocated in the DCE, it triggers also another replacement. In this case the oldest block of the set in the DCE is evicted (Action 5), this is A1. Finally B5 is sent to cache B and allocated where the invalidated block was.

The right part of the figure shows the final state of caches after requesting blocks A6 and B6 for both configurations. It is clear from the result that in the distributed version cache blocks are closer to the requesting node, improving access latency. We can also see that the distributed version also keeps all the newer blocks in the cache, reducing the number of off-chip accesses. This is because the DCE may have inherently data from all cores so that the oldest blocks in the CMP from that set are replaced first.

2.3 Differences between Centralized and Distributed Cooperative Caching

The main differences between these proposals are:

- In the centralized version tags are just a copy of their corresponding caches while in the distributed version tags are ordered like a big shared cache in the DCEs. Since tag entries are not restricted to represent only one cache entry, this organization makes a more efficient use of them. Furthermore, the distributed organization does not require to reallocate a tag when a block is spilled or allocated in another cache. It is only necessary to update the tag information, with the energy savings that this implies.

- The number of tags checked per request in the DCE is equivalent to its associativity -independent of the associativity of the L1s and L2s. In the CCE the number of tags checked is #L1*L1 Associativity + #L2*L2 Associativity. This translates directly in a reduction in the energy consumption.

- The DCEs implement a LRU replacement policy that favours a broad view of evicted blocks instead of the individual replacement of private caches in the centralized version. This allows a more efficient use of cache entries.

- The size of the DCE is independent of the sizes of the L1s and L2s while in the CCE the number of tags has to be equal to the number of L1 and L2 cache entries. The coherence protocol of the Distributed Cooperative Caching framework needs to be able to handle DCE replacements.

- The Distributed Cooperative Caching makes use of several coherence engines that can be distributed across the chip. This organization avoids bottlenecks in the on-chip network and allows to parallelize the request handling. This becomes more important as we increase the number of processors on a chip.

Further work done by Chang and Sohi [4] evaluates dynamic configurations to improve fairness and QoS of the Cooperative Caching. Our work, on the other hand, is focused to improve the scalability and the power consumption.

3. EXPERIMENTAL SETUP

We have evaluated our proposed framework with Simics [13], a full-system execution-driven simulator extended with the GEMS [15] toolset that provides a detailed memory hierarchy model. We have added a power model to the simulator based on Orion [23] to evaluate the energy efficiency of our proposal. Our configuration uses simple cores with small primary caches to improve the aggregate thread throughput by a high number of processors [6]. Table 1 shows the values for the most important configuration parameters.

A configuration with 4 processors has also been simulated but has not been included in the results since traditional organizations such as private or shared caches achieve the same performance and are much simpler. We have used the

Parameter	Value
Number Processors	8-16-32
Instr Window/ROB	16/48 entries
Branch Predictor	YAGS
Technology	70 nm
Frequency	4 GHz
Voltage	1.1 V
Block size	64 bytes
L1 I/D Cache	16 KB, 4-way
L2 Cache	512-256-256 KB, 4-way
Network Type	Mesh with 2 VNC
Hop Latency	3 cycles
Link BW	16 bytes/cycle
Memory Bus Latency	250 cycles

Table 1: Configuration Parameters

SPECOMP2001 workload set with the reference input sets. The Distributed Cooperative Caching framework has been compared against traditional organizations such as shared or private last level cache and also against the the centralized Cooperative Caching. In all the tested configurations two levels of cache are used; as well as a MOESI protocol to grant coherence between nodes. All simulations use a local and private L1 cache and a shared/private L2 cache for every processor. Evaluated configurations are:

Shared Memory. This configuration assumes a Non-Uniform Cache Access (NUCA) architecture. L2 cache is physically distributed across the nodes and logically unified. Addresses are mapped to cache banks in an interleaved way to try to distribute requests in the network. L1 and L2 caches are inclusive and the L2 also includes the directory information for the allocated entries. On a L1 miss, the L2 bank corresponding to the address is accessed. If the block is located in another L1 in read-only mode, then it is replicated in the requesting node L1. Otherwise, the owner is invalidated without having to access the off-chip directory. This configuration tries to optimize cache usage and reduce off-chip accesses.

Private Memory. In this design, a L2 cache bank is assigned to every processor. On a L2 cache miss, memory must be accessed to check if the block is shared and to retrieve the data. This configuration makes a very small usage of the on-chip network and tries to optimize the access latency by placing all cache blocks in the local L2.

Cooperative Caching. We have evaluated two different configurations for the previously presented Cooperative Caching framework. The default Cooperative Caching version uses a Coherence Engine capable of doing 2 transactions per cycle. Because of the centralized nature of this configuration, for a high number of nodes this limitation can saturate the CCE. For a fair comparison with the distributed version we have evaluated the performance of a CCE with 4 R/W ports (Cooperative Caching 4T). This configuration, however, increases the tag access energy due to the increased complexity.

Distributed Cooperative Caching. The Distributed Cooperative Caching proposal also has been evaluated with two configurations. Both of them use 1 DCE for each node/ processor with 2 R/W ports and 4-way associativity. The default configuration uses as many tags as the L2, requiring 128k entries for a total L2 of 4 MB. The second configuration uses twice as many entries and is labeled with 2x. This configuration is used to reduce the effect of invalidations and check the degradation in performance they induce. Since the Cooperative Caching uses a coherence engine entry for every sharer and for the sake of a fair comparison the DCEs use Full map directories (Dir-N). The extra cost in bits for each tag in the case of a 32 core CMP with 32 DCEs is one bit per sharer and 4 bits for the DCE state. This means that if we assume 32 processors and a total L2 of 8 MB (i.e. 256KB per processor), each DCE will have a size of 18 KB. We believe the hardware overhead is reasonable and we don't need to invalidate any sharer, reducing the protocol complexity. This organization, however, may limit the scalability for CMPs with more processors. Partial map directories may be an interesting solution for these configurations but are left for future work. We have fully implemented the cache coherence protocol with the DCE invalidation mechanisms. Invalidation implies two extra states in the DCE state machine and one extra state in the cache state machines. Results shown in the next section already include the extra overhead.

3.1 Power Model

An architectural-level power model for all the interconnection network and memory hierarchy has been implemented. The model is derived from Orion [23] for modelling the buffers, crossbars, arbiters and links. We have estimated the capacitances of all this components taking the technology parameters from Cacti [21]. Also we have used Cacti to calculate the dynamic and static energy consumption of all the caches.

The implemented power model has been validated against data of real multiprocessors. We have compared our implementation against power numbers of the MIT Raw chip multiprocessor [11] and the fully place and routed ASIC design of Mullins [17]. Validation results show a relative error of about 10%. We also include a power estimation for the cores based on power values found for similar configurations in the literature [16].

4. RESULTS

4.1 Power/Performance Evaluation

In this section, we are going to present the evaluation of our proposed framework compared to traditional memory configurations and the centralized cooperative caching. Figure 5 shows the speedups of the studied configurations over the shared cache organization. For all the different number of processors our proposal outperforms the other configurations. In addition, as we increase the number of processors we can see that the improvement over the Cooperative Caching broadens.

Performance results show that the speedups for the art and applu benchmarks are very high for all configurations. This is because the performance for the shared memory configuration is heavily penalized by network congestion. In these benchmarks, the number of misses/instruction of the first level cache is very high and motivates an increase in the network usage for accessing the shared L2.

The weight of the power consumption of the cache subsystem and the interconnect may vary depending on the core configuration. For a small number of complex cores, the power consumption of cores will be significant. However,

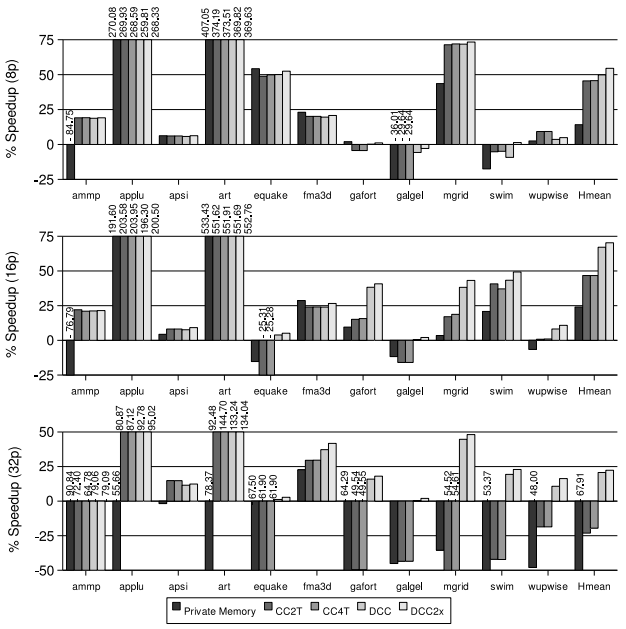

Figure 5: Normalized speedup over Shared Memory Configuration.

for a big number of simple cores the influence of the interconnect in the overall power consumption will be bigger. For that reason we have found interesting to evaluate the power/performance relation for the caches and interconnect alone, depicted in Figure 6. We have also evaluated this relation considering the core energy consumption as we will see in short.

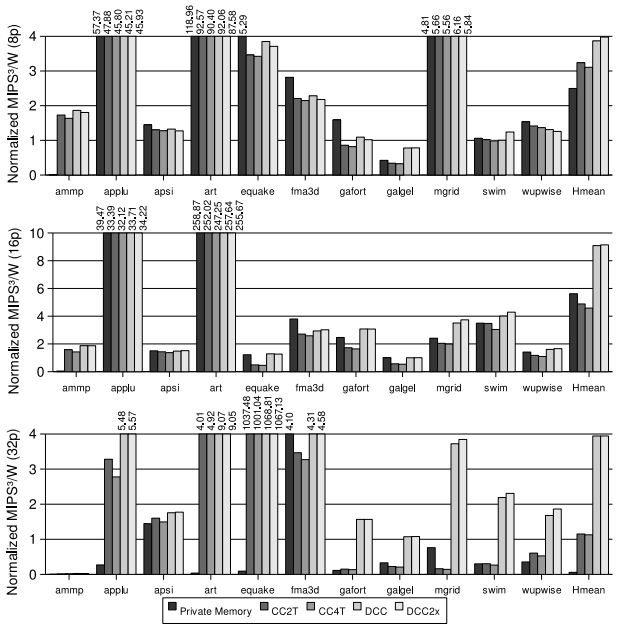

Figure 6: Normalized MIPS^3/W over Shared Memory Configuration.

Results show that private caches and more complex schemes have a similar power/performance relation for a reduced number of cores. For a bigger number of processors, however, the number of off-chip misses degrades the performance of private caches dramatically. Other configurations like shared caches and Cooperative Caching show a significant increase of power consumption. This increase is due to the high amount of network traffic in the Shared cache configuration and due to the energy consumption of the Coherence Engine in the CC scheme.

We can also see in Figure 6 that the extra number of ports in the Cooperative Caching 4T configuration does not provide a power/performance improvement. This configuration has a small increase of performance for some benchmarks but also increases the power consumption due to the increased complexity. For these configurations we can see that the Distributed Cooperative Caching scheme is the most energy efficient.

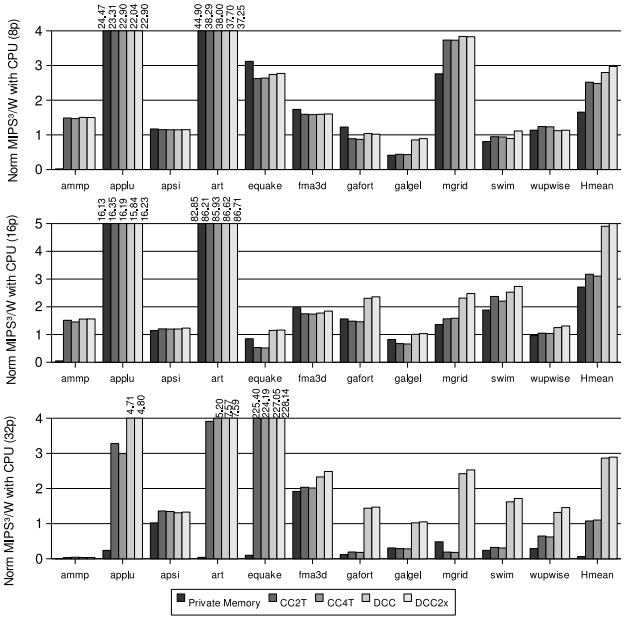

Figure 7: Normalized MIPS^3/W over Shared Memory Configuration with CPU power.

Figure 7 shows the power/performance relation considering the CPU power. In this case variations in the energy consumption of the memory subsystem have a smaller influence in the results, reducing the differences between configurations. The Distributed Cooperative Caching, however, remains as the most energy efficient solution for a 32-core CMP. It shows an average improvement of 2.61x over a traditional shared memory configuration and a 3.33x over the Cooperative Caching framework.

Two interesting parameters for evaluating the memory hierarchy of chip multiprocessors are shown in Figures 8 and 9. These are the number of off-chip misses and the average L1 miss latency. Traditional configurations such as shared or private caches only achieve good results in one of them. The best results regarding off-chip misses are obtained by the shared cache configuration while the private configuration shows a better average latency.

On the other hand, hybrid proposals like the Centralized and the Distributed Cooperative Caching try to improve both parameters by locating cache blocks in the local nodes but also making use of all the on-chip cache space.

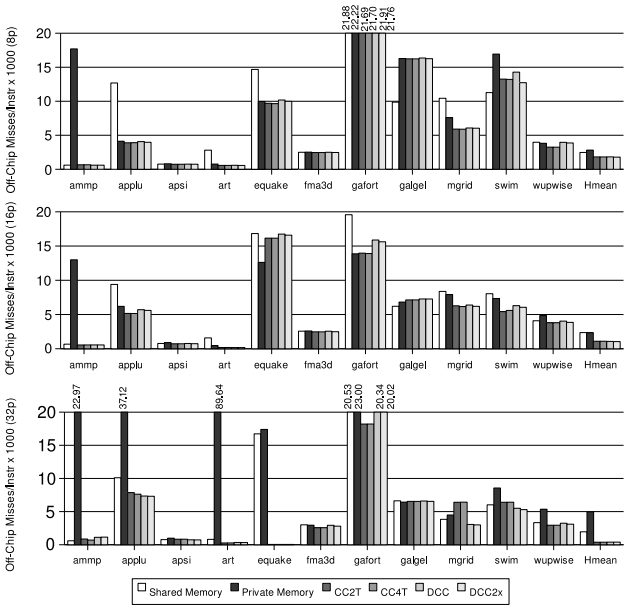

Figure 8: Off-Chip Misses per thousand instructions.

The Distributed Cooperative Caching has a number of off-chip misses close to the shared cache configuration due to its replacement policy. The shared tag structure of the DCEs invalidates blocks on a replacement through a Least Recently Used (LRU) policy. On the other hand, in the centralized organization, replacement in caches is done independently with the least recently used block of each cache. With the latter, spilled blocks are usually newer in the local node and generate evictions of other entries that may be more recent, as seen in the example of section 2. The number of off-chip misses, however, remains very similar to the Distributed Cooperative Caching.

Speedup results of Figure 5 also show that the performance of the private configuration is highly penalized for a big number of processors. The explanation of this behaviour is shown in Figure 8, where we can see that the number of off-chip misses is significantly increased due to an inefficient use of the last-level cache.

The average L1 miss latency, depicted in figure 9, shows that CC and DCC achieve the best results. This is because both configurations behave similarly to a private cache configuration and improve it further by adding a sharing mechanism that reduces off-chip accesses.

We can also see that the network congestion produced in some benchmarks for the shared configuration leads to a very high miss latency. This results explain the degradation in performance that leads to very high speedups for the other configurations.

4.2 Sensitivity Studies

As said in section 3, an extra configuration for the Cooperative Caching framework with more ports has been added

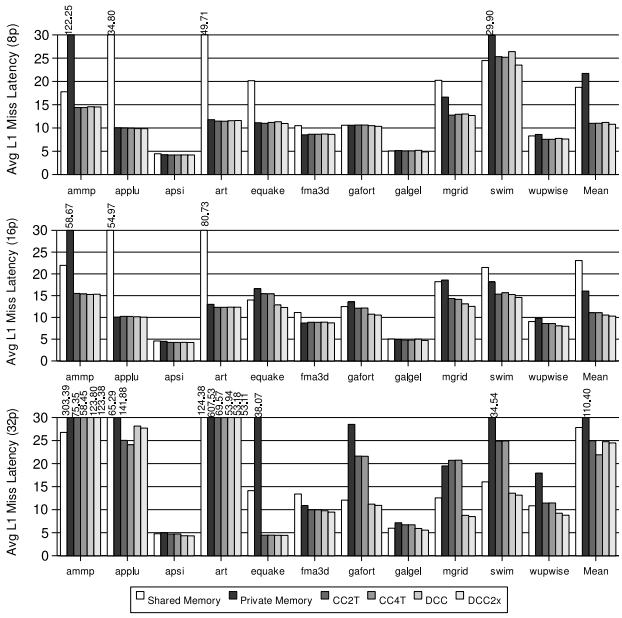

Figure 9: Average L1 Miss latency.

to see if saturation of the CCE could be avoided for a high number of processors. Figure 5 shows that the performance of the configuration with 4 read/write ports does not improve significantly. Results for this configuration are even worse in the MIPS^3/W relation due to the higher power consumption added by the extra ports.

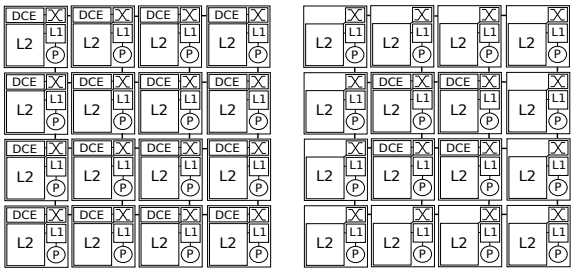

Figure 10: DCC16CE and DCC4CE organization.

We have also conducted a study to exploit the configuration flexibility of the DCE. The behaviour of a system with a DCE for each node and 16 processors (DCC16CE) has been compared with a system with 4 DCEs and 16 processors (DCC4CE). Figure 10 shows the DCE distribution of both configurations. In the 4 DCE version the number of tags has been increased so both configurations have the same number of entries. Also, three different associativities for the DCEs have been evaluated with our framework, 4-way (4A), 8-way (8A) and 16-way (16A) DCEs. Figure 11 shows the speedups and power/performance relation of all these configurations over the base DCC configuration.

Results show that configurations with higher associativity achieve a slightly better performance. This is because the number of replacements per request is reduced. However, the number of tags compared in the DCEs depends on its associativity. The power/performance relation shows that

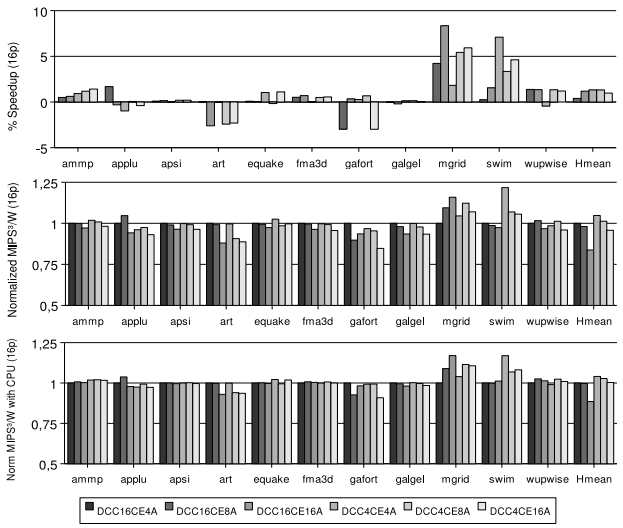

Figure 11: DCC Optimal Configuration Study.

the speedup obtained for a higher associativity is not enough to compensate the additional power requirements. On the other hand, the usage of less DCEs than nodes increases the power/performance relation by 5%. This improvement is explained by the reduction in the average distance to the DCE. A balanced solution between distance and request distribution across the network needs to be chosen for every case. The DCC scheme, however, provides a flexible framework to find the optimal configuration.

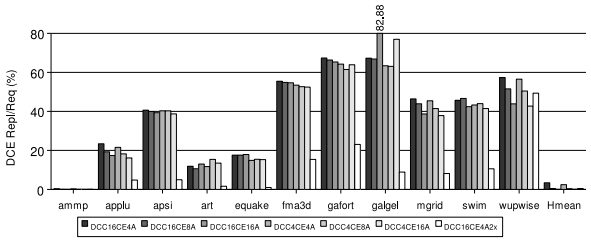

Figure 12: DCE replacements per request.

Figure 12 shows the percentage of requests that end up in an invalidation of a cache block due to the lack of tags. For the base configuration almost half of the requests to the DCEs end up with an invalidation, except for the ammp benchmark that is very cpu intensive and does not stress the memory system. These invalidations may cause a degradation of the overall performance, so we have evaluated our Distributed Cooperative Caching framework with twice the initial number of entries (labeled 2x in the figure). This configuration may be to expensive in hardware for a real implementation but allows us to see how much performance is lost. We can see that the number of replacements per request is highly reduced and this is translated in a performance improvement. This improvement, however, is not very high since evicted blocks of the original configuration are always the least recently used from that set.

5. RELATED WORK

Memory hierarchy for multiprocessors has been widely studied in the last decade to improve performance and scalability of cc-NUMA architectures [1, 12]. However, the different latency and bandwidth constraints of CMP networks generates new research challenges to find the best memory organization. Many studies have appeared last years in this field [2, 3, 4, 5, 10, 18, 24]. Proposals are divided between snoop and directory based.

CMP-NuRapid [5] from Chisti et al. is one of the snoop based coherence schemes. This work also proposes a duplication of tags in each node. In this scheme, tags are copied to the local node tag set the first time the block is accessed, and the data replicated in a closer cache if the block is accessed again. This way, all subsequent accesses will have a smaller latency. It has the advantage that several blocks of the same set can be in the closer cache if they are used often with no risk of being replaced since tags and data are separated. This proposal has the power and performance limitation of requiring most of the times a transfer of the least used block to a slower group when we want to add a new one to the fast and is not scalable because blocks are found via snoop requests. Another approach that uses a snoopy scheme to implement cache coherence is the Uncorq protocol [20]. This proposal uses a logical unidirectional ring embedded in a 2D Torus. The ring is only used by control messages while the rest can use any path. This mechanism falls into the same problems as the previous one and it achieves only good performance for a small number of nodes. The scheme proposed by Martin et al. [14] tries to separate performance from correctness. This idea tries to optimize the protocol for common cases but rely on a correctness substrate to resolve races. Also a snoop-based coherence protocol is used, limiting the scalability.

On the other hand, several proposals use a directory based protocol like in our case. One of them is the Cooperative Caching framework [3] that has been widely described previously and that has been extended to provide a better fairness and QoS [4]. The Utility-Based Cache Partitioning [18] also is a directory based configuration and uses a big unified 16-way cache. In this cache, ways are assigned to nodes according to the benefits that can produce to each thread. Dybdahl et al. [9] proposed a similar technique but with a different selection criteria for the sharing mechanism. Both proposals do not try to reduce latency by allocating blocks in the closer nodes and are not scalable due to the centralized nature of the last level cache. Another hybrid proposal is the Victim Replication [24] protocol. This configuration has a traditional distributed shared memory but adds a new replacement mechanism to reduce the miss latency. By default, blocks have a fixed L2 cache for being stored but in L1 replacements the block is replicated in the local cache if there is a spare place. The main limitation of this configuration is that under heavy load conditions behaves as a normal shared cache configuration. Finally NUCA Substrate [10], proposes a shared pool of small cache banks that can have different degrees of sharing. Dynamic mapping allows data to be stored in multiple banks but requires a tag check of all the possible destinations. Results show that statically mapping banks has similar performance and much less complexity.

In addition to all these configurations, Beckmann et al. proposed the Adaptive Selective Replication mechanism [2].

This method can be used in all the previous configurations with private L1s that replicate blocks on their evictions like our framework and tries to optimize the level of replication dynamically.

Our work, as seen before, is classified in the directory based schemes. These protocols achieve a better scalability and we intend to design a scheme for multiprocessors with a large number of cores. While most of the proposals use either a centralized cache or a centralized tag structure, our work tries to distribute coherence messages across the on-chip network to avoid bottlenecks like a cc-NUCA organization and keep the average miss latency of private caches.

6. CONCLUSIONS

While for a reduced number of nodes traditional configurations like shared or private caches provide the best power/performance relation, in the advent of the many core era it is essential to devise a more efficient solution. We have seen how the Distributed Cooperative Caching framework provides a scalable and energy efficient organization for large multicore architectures using less hardware resources than the centralized version. For a 32-core CMP, execution time is improved by a 21% over a traditional shared memory configuration and by 57% over Cooperative Caching. When we also consider the energy, results are even better, showing a increase in the $MIPS^3/W$ of 3.66x over a traditional shared memory configuration and a 4.30x over the Cooperative Caching framework.

7. ACKNOWLEDGMENTS

We would like to thank Antonio González for his comments on this work. We also would like to thank Hangsheng Wang and Li-Shiuan Peh from the Orion group for providing power numbers of the MIT Raw microprocessor and the people of the Wisconsin Multifacet Project for their help in the use of GEMS. This work has been supported by the Generalitat de Catalunya under grant 2005SGR00950, the Spanish Ministry of Education and Science under grants TIN2004-03072 and TIN2007-61763 and Intel.

8. REFERENCES

[1] M. Acacio, J. Gonzalez, J. Garcia, and J. Duato. A new scalable directory architecture for large-scale multiprocessors. In *HPCA '01: 7th International Symposium on High-Performance Computer Architecture*, pages 97–106, January 2001.

[2] B. Beckmann, M. Marty, and D. Wood. Asr: Adaptive selective replication for cmp caches. In *MICRO-39: 39th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2006.

[3] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *ISCA '06: 33rd Annual International Symposium on Computer Architecture*, pages 264–276, June 2006.

[4] J. Chang and G. S. Sohi. Cooperative cache partitioning for chip multiprocessors. In *ICS '07: 21st Annual International Conference on Supercomputing*, pages 242–252, June 2007.

[5] Z. Chishti, M. Powell, and T. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache architectures. In *MICRO-36: 36th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 55–66, December 2003.

[6] J. Davis, J. Laudon, and K. Olukotun. Maximizing cmp throughput with mediocre cores. In *PACT '05: 14th International Conference on Parallel Architectures and Compilation Techniques*, pages 51–62, September 2005.

[7] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core opteron processor. In *ISSCC '07: IEEE International Solid-State Circuits Conference*, pages 102–103, February 2007.

[8] P. Dubey. A platform 2015 workload model: Recognition, mining and synthesis moves computers to the era of tera. *Intel White Paper, Intel Corporation*, 2005.

[9] H. Dybdahl and P. Stenstrom. An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors. In *HPCA '07: 13th International Symposium on High Performance Computer Architecture*, pages 2–12, February 2007.

[10] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. Keckler. A nuca substrate for flexible cmp cache sharing. In *ICS '05: 19th Annual International Conference on Supercomputing*, pages 31–40, June 2005.

[11] J. S. Kim, M. B. Taylor, J. Miller, and D. Wentzlaff. Energy characterization of a tiled architecture processor with on-chip networks. In *ISLPED '03: International symposium on Low power electronics and design*, pages 424–427, August 2003.

[12] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the dash multiprocessor. In *ISCA '90: 17th Annual International Symposium on Computer Architecture*, pages 148–159, May 1990.

[13] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, 2002.

[14] M. Martin, M. Hill, and D. Wood. Token coherence: decoupling performance and correctness. In *ISCA '03: 30th Annual International Symposium on Computer Architecture*, pages 182–193, June 2003.

[15] M. Martin, D. J. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, 2005.

[16] M. Monchiero, R. Canal, and A. Gonzalez. Power/performance/thermal design space exploration for multicore architectures. *IEEE Transactions on Parallel and Distributed Systems*, 19(5):666–681, May 2008.

[17] R. Mullins. Minimising dynamic power consumption in on-chip networks. *International Symposium on System-on-Chip*, pages 1–4, November 2006.

[18] M. Qureshi and Y. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches.

In *MICRO-39: 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 423–432, December 2006.

[19] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs. The implementation of the 65nm dual-core 64b merom processor. In *ISSCC '07: IEEE International Solid-State Circuits Conference*, pages 106–590, February 2007.

[20] K. Strauss, X. Shen, and J. Torrellas. Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors. In *MICRO-40: 40th Annual IEEE/ACM International Symposium on Microarchitecture*, December 2007.

[21] D. Tarjan, S. Thoziyoor, and N. Jouppi. Cacti 4.0. Technical report, HP Labs Palo Alto, June 2006.

[22] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *ISSCC '07: IEEE International Solid-State Circuits Conference*, February 2007.

[23] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *MICRO-35: 35th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 294–305, November 2002.

[24] M. Zhang and K. Asanovic. Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *ISCA '05: 32nd Annual International Symposium on Computer Architecture*, pages 336–345, June 2005.