APPENDIX

A. 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 and store information about the owners. 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.

- 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 

\[ \#L1 \times L1 \text{Associativity} + \#L2 \times L2 \text{Associativity}, \]

which leads to a reduction of the energy consumption.

- The DCEs implement a LRU replacement policy that favors 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. Therefore, the Set Bits selecting the Coherence Engine entry in CC are the same ones that are used in the L1 and L2 caches. Therefore, for 16KB 4-way L1 and 256KB 8-way caches, a different number of bits is going to be used in the CCE for the L1 (6 bits) and the L2 (9 bits) entries. On the other hand, the number of bits used in the DCEs depends on the number of DCE entries, the number of DCEs and their associativity. Therefore, if we use as many DCE entries as L2 cache entries and as many DCEs as nodes (16), we are going to use 4 bits to map the DCEs as CE Bits) and 9 bits to map the different sets (Set Bits). The coherence protocol of the Distributed Cooperative Caching framework also 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 can handle requests in parallel. This becomes more important as we increase the number of processors on a chip.

- The hardware overhead of DCC compared to CC is the storage area used to keep track of the sharers in each tag. DCEs use Full map directories (Dir-N), which require one bit per sharer. This means that if we assume 32 processors and a total L2 of 8 MB (i.e. 256KB per processor), the overhead per DCE would be 16KB. 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.

B. The DCE replacement policy: an example

To show the benefits of the DCE tag replacement policy, Figure 15 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 behavior 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 (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 space, B5 is spilled to node A. Block B1 is evicted from the chip since subsequent spillings are not allowed.

In the bottom part of Figure 15 the behavior 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. The Distributed Cooperative Caching, however, does not enforce actively a local allocation. In the DCC example cache blocks are closer to the requesting node thanks to the replacement mechanism of the coherence engines, which may have inherently data from all cores. Replacements in the coherence engine entries evict oldest blocks avoiding them to be spilled when evicted, which would force an eviction in the destination cache of a newer block. This effect, however, has a limited impact if coherence engines have a reasonable number of entries. Distance-Aware Spilling, on the other hand, takes into account distances and enforces closer reallocations.

C. Distance-Aware Spilling Average Distance

In this section the distance of the destination nodes is shown for the Distance-Aware Spilling using mesh and ring networks. Figure 16 shows the average number of nodes at each distance for the random and the two Distance-Aware spilling policies. The figure also shows the average distance for these configurations. We can see that the average distance is 1 hop when using 2 destinations (DAS2n) and 1.5 hops when using 4 (DAS4n), while for a random destination selection the average distance is 2.7 hops.

In the case of the ring network, the benefit of Distance-aware spilling is much higher since the average node distance is 4.27 for the random distribution (as shown in Figure 17). Therefore, these types of topologies can benefit more of a Distance-aware spilling.

D. Experimental Setup

We have evaluated the presented configurations with Simics [17], a full-system execution-driven simulator extended with the GEMS [19] toolset that provides a detailed memory hierarchy model. We have added a power model to the simulator based on Orion [30] 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 I shows the values for the most important configuration parameters.

We have used all the Spec OMP 2001 workload set with the reference input sets for uniprogrammed configurations and ten pairs of these benchmarks for the multiprogrammed configurations. In multiprogrammed configurations each benchmark runs in half of the processors. Threads are allocated together so one half of the die runs the first benchmark and the other half the second. Finally we have also studied another set of multiprogrammed benchmarks combining multithreaded Spec OMP 2001 applications with single threaded Spec CPU 2006 applications.

Benchmarks have been characterized by the number of spilled blocks per instruction, number of reused blocks, and reuse per-

![Fig. 16. Average distance to destination nodes in a mesh network.](image1)

![Fig. 17. Average distance to destination nodes in a ring network.](image2)

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

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Spill/Instr</th>
<th>Reuse</th>
<th>Spill Reuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swim</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Gafort</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Ammp</td>
<td>Low</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Art</td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Apsi</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>444_Namd</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>450_Soplex</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>456_Hmmer</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>459_GemsFDTD</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

Table II: Simulated Benchmarks in multiprogrammed configurations.
percentage of spilled blocks. Then benchmarks of table II have been selected to have all possible types of behaviors.

All the Distributed Cooperative Caching configurations have 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.** (CC) This configuration evaluates the Cooperative Caching framework [3]. The default Cooperative Caching version uses a Coherence Engine capable of doing 2 transactions per cycle.

**Adaptive Selective Replication.** (ASR) This configuration evaluates the Adaptive Selective Replication [2] (Only for the last benchmark set). This configuration uses 8 K entry 8 way NLHBs and 512 entry 8 way VTBs for each node and coherence is granted through a distributed directory.

**Distributed Cooperative Caching.** (DCC) 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 8-way associativity. The default configuration uses as many tags as the L2, requiring 64k entries for a total L2 of 4 MB. The second configuration uses twice as many entries as the L2, requiring 64k entries for a total L2 of 4 MB. The default configuration makes a very small usage of the on-chip network and memory hierarchy has been implemented. The model has been validated against data of real multiprocessors.

**Distance-Aware Spilling** A configuration of the DCC mechanism with distance aware spilling to 2 neighboring nodes (DCC_DAS2n) and to 4 neighboring nodes (DCC_DAS4n).

**Selective Spilling** (DCC_SSS) with Cₛ of 100k cycles and a spilling threshold of 5 reused blocks.

**Power-Efficient Spilling** (DAS4n_SSS) A configuration with the Distance-Aware spilling (4n) and the Selective spilling together.

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 16 core CMP with 16 DCEs is one bit per sharer and 4 bits for the DCE state. This means that if we assume 16 processors and a total L2 of 4 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. A configuration tries to optimize cache usage and reduce off-chip accesses.

An architectural-level power model for all the interconnection network and memory hierarchy has been implemented. The model is derived from Orion [30] for modeling the buffers, crossbars, arbiters and links. We have estimated the capacitances of all this components taking the technology parameters from Cacti [28]. 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 [14] and the ASIC design of Mullins [22]. 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 [21].

**E. DCE Size Sensitivity Studies**

We have also conducted a study to exploit the configuration flexibility of the DCE. The behavior 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 18 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. In addition, three different associativities for the DCEs have been evaluated with our framework, 4-way (4A), 8-way (8A) and 16-way (16A) DCEs. Figure 19 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 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 20 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 too expensive in hardware for a real implementation but it 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.

F. Multiple Multi-threaded Benchmarks Evaluation

To see the interaction between applications and the possible inter-thread interferences we have evaluated several multipro-
programmed benchmarks. These benchmarks combine pairs of Spec OMP applications, executing each one with 8 threads.

1) Mesh Network: Figure 21 shows the performance, energy efficiency and network activity for the studied multiprogrammed set of benchmarks. The presented energy-aware techniques effectively cut down network usage without degrading performance. Distance Aware Spilling achieves a reduction of 14.2% for the DAS2n and 10.8% for the DAS4n. In the case of Selective Spilling, network usage is further reduced up to 20.8% since the amount of spilling is limited to reused data. The combined solution shows a reduction of 26.6% of the network traffic. All these improvements are achieved while keeping the same performance of the DCC with random spilling (the best performing configuration).

The second plot of Figure 21 shows both the benefits of the performance increase and the reduction in network usage (power). Distributed Cooperative Caching also outperforms other configurations by a 22-31% in energy efficiency. The addition of Distance-aware spilling, pushes up the savings an extra 4% (DAS2n). This improvement is achieved although network power consumption is only a small part of the total power. Networks of next generation tiled microarchitectures, however, are expected to have a greater importance in the overall performance and power, leading to greater impact of these techniques.

2) Ring Network: In ring networks, as it was shown before, the penalty of accessing far nodes is higher. As it will be shown, this limitation is reduced in the Distance-Aware Spilling techniques which directly translates to better performance. Results are normalized to the same DCC organization running on top of the ring network. Therefore Distance-Aware spilling is more suitable for this kind of configurations. This can be seen in the first graph of Figure 22 where DAS2n achieves a performance improvement of 4.24% and DAS4n of 6.26%. Configurations with the Selective Spilling technique keep the same performance as the random configuration but, as it can be seen in the third graph, reduce the network usage by 21.7%. On the other hand, Distance-Aware (DAS) configurations achieve a reduction of 16.7% for the DAS2n and 14.1% for the DAS4n while increasing the overall performance. If the benefits of both configurations are combined, the reduction of the network traffic is even higher, reaching a 30%.

Finally, the energy efficiency of the evaluated configurations can be seen in the second graph of Figure 22. The performance increase and network activity reduction of the Distance-Aware techniques is translated in an improvement of the energy efficiency of a 11.5% for the DAS2n and a 16.4% for the DAS4n. Once more, the Selective Spilling configurations, do not show a big variation with respect to the random spilling since the power contribution of the on-chip network in the overall power is not very high. The energy efficiency of the shared cache configuration is very low due to the intensive use of the on-chip network, that in this case is aggravated by the lower capacity of the network (when compared to a mesh). Private caches, on the other hand, show a good energy efficiency due to the low network usage but have a much higher number of off-chip misses that reduce its performance. This would probably increase the overall power had we considered the memory controller energy consumption.

3) Spilling Reuse: Figure 23 shows the average distance of reused blocks. As expected, the average reuse distance is significantly reduced for distance aware spilling techniques which explains the network usage reduction. This distance, however, is slightly higher than the spilling distance since nodes can reuse data spilled by other processors.

Finally, Figure 24 shows the percentage of spilled blocks that are reused later. It can be seen that random spilling makes very small reuse since all evicted blocks are spilled. On the other hand, Selective Spilling is able to double the average reuse in both networks by spilling only data from nodes with reuse and achieves up to 29.5% reuse in the mesh for the art_anmp benchmarks. The combined configuration (DAS+SS) also improves the random configuration in both cases while reducing the distance to the destination nodes. DAS techniques reduce the distance to destination nodes in exchange of available cache space. Therefore spilled blocks are evicted earlier when the cache usage is unbalanced and reuse is reduced.

G. Multiple Single and Multi-threaded Benchmarks Evaluation

In this last section we can see the behavior of the evaluated techniques with pairs of single threaded Spec CPU applications combined with one Spec OMP application. Each Spec CPU application is replicated 4 times and the Spec OMP application is executed with 8 threads, making a full usage of the 16 nodes. This combination of benchmarks makes a much more intensive usage of the memory hierarchy and allows to see the behavior of the different configurations in high demand situations.

To complement the evaluation of DCC in this section we have also modeled the Adaptive Selective Replication (ASR) and compared the performance of both configurations. The evaluated version of ASR also makes use of DCEs to provide coherence in order to have a decentralized structure.
Fig. 22. Normalized performance, energy efficiency and network usage over DCC, Random Ring.

1) Mesh Network: Figure 25 shows the behavior of Distributed Cooperative Caching with single and multi-threaded applications.

It can be seen that Cooperative Caching behaves like traditional shared and private configurations. This is due to the high usage of the memory hierarchy which saturates the centralized directory (CCE). ASR, on the other hand, has a good performance which in some cases outperforms DCC but in the most memory intensive configurations (those using ammp and 456_Hmmer) does not perform well. This is explained because most of the applications do not share data and, therefore, in such cases private caches are desirable. ASR is implemented as a distributed shared cache which under low utilization all cores can replicate data in the neighboring caches. Under high utilization, however, data cannot be replicated to reduce off-chip misses and, therefore, must be accessed in far caches. This also explains the high network usage of the ASR configuration in the third plot of Figure 25. DCC, in such cases behaves as a system with private caches with lower latency and lower network usage. This allows to get an average performance improvement of 43% over shared caches, 47% over private caches, 58% over Cooperative Caching and 12% over ASR. The non-sharing behavior of single-threaded applications benefits the local data allocation of DCC. The power-efficient techniques, however, do not improve performance over random spilling. As in previous configurations, private caches, show a good performance for this network topology but DCC with Distance-aware spilling to 2 nodes outperforms them by 33% and improves energy-efficiency 55%.

H. Related Work

This section presents the previous work related to this paper. Proposals have been divided between snoop and directory based. CMP-NuRapid [5] from Chishti 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.
Fig. 25. Normalized performance, energy efficiency and Network Usage over DCC_Random Mesh.

Fig. 26. Normalized performance, energy efficiency and network usage over DCC_Random Ring.
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 [27]. 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. [18] 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 the ones being studied in this paper [3], [10], [11] that have been widely described previously. The Utility-Based Cache Partitioning [23] 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. [8] 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 [31] 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 [13], 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, several proposals have appeared that dynamically adapt their sharing policies like the Cooperative Cache Partitioning [4] that tries to provide a better fairness in the cache assignment or the Elastic Cooperative Caching [12], more focused on reducing off-chip misses. Several dynamic proposals also have worked on optimizing NUCA organizations for chip multiprocessors [9], [16], [20]. More recently Qureshi [24] has also proposed a dynamic technique that spills blocks to nodes in order to reduce the overall cache misses. This technique, however, is based on a snoop protocol and is only useful for a small number of processors. Srikantaiah et al. [26] presented the Adaptive Set Pinning, a technique to reduce inter-processor misses by assigning a replacement ownership to every set. This ownership is varied dynamically to optimize the cache usage. 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.