DAPSCO: Distance-Aware Partially Shared Cache Organization

ANTONIO GARCÍA-GUIRADO and RICARDO FERNÁNDEZ-PASCUAL,
Universidad de Murcia
ALBERTO ROS, Universidad de Murcia and Universidad Politécnica de Valencia
JOSÉ M. GARCÍA, Universidad de Murcia

Many-core tiled CMP proposals often assume a partially shared last level cache (LLC) since this provides a good compromise between access latency and cache utilization. In this paper, we propose a novel way to map memory addresses to LLC banks that takes into account the average distance between the banks and the tiles that access them. Contrary to traditional approaches, our mapping does not group the tiles in clusters within which all the cores access the same bank for the same addresses. Instead, two neighboring cores access different sets of banks minimizing the average distance travelled by the cache requests. Results for a 64-core CMP show that our proposal improves both execution time and the energy consumed by the network by 13% when compared to a traditional mapping. Moreover, our proposal comes at a negligible cost in terms of hardware and its benefits in both energy and execution time increase with the number of cores.

Categories and Subject Descriptors: B.3.2 [**Memory Structures**]: Design Styles—*Cache memories*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)

General Terms: Performance

Additional Key Words and Phrases: Last-level cache, static block mapping, distance to cache, cache, power

ACM Reference Format:
García-Guirado, A., Fernández-Pascual, R., Ros, A., and García, J. M. 2012. DAPSCO: Distance-aware partially shared cache organization. ACM Trans. Architec. Code Optim. 8, 4, Article 25 (January 2012), 19 pages.
DOI = 10.1145/2086696.2086704 http://doi.acm.org/10.1145/2086696.2086704

1. INTRODUCTION AND MOTIVATION

Tiled CMPs are regarded as a good design for many-core CMPs due to the low complexity of the design compared with other alternatives and the ease to scale up the number of cores simply by means of adding more tiles. In most of these proposals, there is a last-level cache (LLC) which is physically distributed among the tiles. Different LLC organizations are possible to avoid costly off-chip accesses [Liu et al. 2004] and to optimize on-chip communication and cache utilization. These organizations range from a completely shared LLC [Kim et al. 2002; Kongetira et al. 2005; Held and Koehl 2006; Kalla et al. 2010] in which all cache banks of the chip can be accessed by any core to private LLCs [amd 2007; Agny et al. 2010] in which each core accesses its own private

This work was jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 00001/CS/2007, by the Generalitat Valenciana under Grant PROMETEO/2008/060, and also by the Spanish MEC and European Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2009-14475-C04-02". A. García-Guirado was also supported by a research grant from the Spanish MEC under the FPU National Plan (AP2008-04387).

Authors' addresses: A. García-Guirado, R. Fernández-Pascual, A. Ros, and J. M. García, Departamento de Ingeniería y Tecnología de Computadores, Universidad be Murcia; A. Ros, Departamento de Informática de Sistemas y Computadores, Universidad Politécnica de Valencia.

DOI 10.1145/2086696.2086704 http://doi.acm.org/10.1145/2086696.2086704

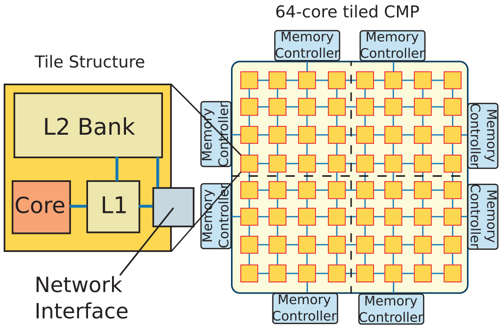


Fig. 1. Diagram of a 64-core tiled CMP with a sharing degree of 16. The dashed lines separate the 4 clusters of tiles which share LLC banks among them.

cache bank, with intermediate partially shared cache organizations in between [Nayfeh et al. 1996; Huh et al. 2005; Hammoud et al. 2009]. We will use the term *sharing degree* to denote the number of cores that can access each cache bank (this will be one for private LLC organizations and will be equal to the number of cores in a totally shared LLC organization). Equivalently, the sharing degree is also the number of different banks that can be accessed by each core. It is not yet clear which is the optimal sharing degree, but most proposals for many-cores assume at least some degree of sharing in the LLC.

In a CMP with a shared or partially shared LLC organization, the time and energy that a request and its response spend in the interconnection network is a significant part of the total time and energy spent accessing the LLC. Both time and energy depend on the distance from the core that accesses some data to the particular bank that holds that data.

Logically, increasing the sharing degree increases the average distance that needs to be travelled by the messages because some of the LLC banks will have to be further from the core. Hence, as the number of cores in CMPs increases, totally shared organizations become less attractive in this regard. However, some degree of sharing is still desirable because it improves LLC utilization. This is because shared caches allocate a single copy of each shared data for all cores, contrary to private caches which replicate shared data in the private caches of every core that accesses these data. This replication increases cache pressure and results in extra LLC misses which in turn increase off-chip traffic. Since off-chip bandwidth increases more slowly than the number of cores in a CMP [Rogers et al. 2009], avoiding LLC misses becomes even more important as the number of cores increases. Hence, partially shared organizations provide a good compromise between the fast latency of private caches and the improved capacity utilization of shared caches.

The most usual way to organize partially shared organizations is by dividing the CMP in clusters of tiles which share their banks of LLC among them (like in Figure 1). The mapping of memory blocks to LLC banks is usually performed by looking at some address bits (particularly, $log_2 sd$ bits, where sd is the sharing degree). By using this static mapping, the set of banks accessed by cores in two different clusters do not overlap. Additionally, a directory is used to enforce cache coherence among the clusters.

The above organization does not take into account the distance between a core and the cache banks it accesses. In fact, there are very significant differences between cores within the same cluster, because the cores in the center of clusters will have the LLC banks that they need to access nearer (on average) than the cores near the edges of the clusters. Other organizations have been proposed (see section 6), including dynamic mappings.

In this paper, we propose DAPSCO (Distance-Aware Partially Shared Cache Organization). DAPSCO uses a static block mapping policy that optimizes the average distance to access remote LLC banks, improving the energy consumption and the overall performance of the system. This is possible because, differently from other static mappings, DAPSCO mappings do not group tiles in clusters. Instead, each LLC bank serves a portion of the memory space to its neighboring cores, and every core accesses a different set of LLC banks.

Since DAPSCO employs an static mapping, it does not need any extra power-consuming structures or mechanisms with respect to traditional mapping policies for partially shared caches. Hence, no trade-off between energy and performance exists, and both execution time and energy consumption are improved. As a result, the overall performance of a 64-core CMP increases by 13% and the energy consumption of the network also improves by 13%. We predict even larger improvements in CMPs with more cores.

Finding the optimal block mapping that minimizes the number of links traversed for a given on-chip network topology is an intractable problem for large CMPs, and therefore we use a greedy algorithm to find near optimal block mappings for both mesh and torus topologies. It is important to notice that this algorithm is only employed to obtain the static mapping function for each CMP configuration.

The rest of this paper is organized as follows. Section 2 gives some background on the target architecture. Section 3 describes DAPSCO and how to find near-optimal DAPSCO mappings. Section 4 evaluates DAPSCO compared to current block mapping policies for partially shared caches. In Section 5 we show that DAPSCO is orthogonal to mechanisms based on partially shared LLCs by applying DAPSCO to Reactive NUCA [Hardavellas et al. 2009]. Section 6 discusses some related work. Finally, our conclusions can be found in Section 7.

2. BACKGROUND

The base architecture targeted by our proposal is a tiled-CMP like that shown in Figure 1, in which the chip is built by replicating basic building blocks named tiles. Each tile contains a processing core, separate private L1 instruction and data caches, one bank of the last level cache (LLC), which in our case corresponds to the L2 cache, and a network interface to communicate with other tiles. The L1 and L2 caches are non-inclusive. Clusters of cores share their L2 banks, as shown in Figure 1. An optimized MOESI, directory based cache coherence protocol is used. We use 64 tiles through the paper for all examples and for the evaluation section, although our proposal would achieve higher benefits with a larger number of cores. In the rest of the paper, we call this design “traditional partially shared cache organization”.

As mentioned before, the *sharing degree* of the cache determines the number of cores that access each cache bank (which is equal to the number of banks accessed by each core) and, in the case of traditional partially shared caches, this also determines the size of the clusters of tiles that share the same cache banks. We always assume the same sharing degree for every LLC bank (i.e. all LLC banks are accessed by the same number of cores) for the sake of fairness in the pressure exerted over them.

In this design, certain bits of the memory address of each block (bank selector) determine which tile contains the home cache bank for that block. Figure 2 shows how the cores of the upper left quadrant use the “bank selector” bits of an address to determine the LLC bank containing the block and its associated directory information. The correspondence between all the possible values of the bank selector and the corresponding home cache banks of the cluster is also shown in the figure. Notice the shades under the bit values that indicate the correspondence between bits in the bank selector and

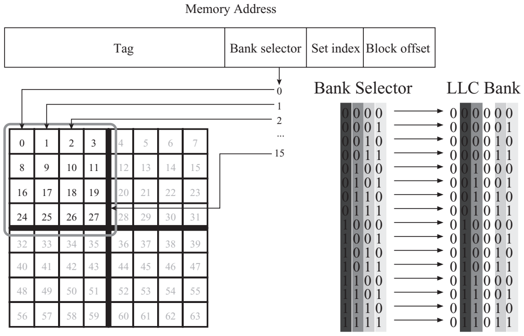


Fig. 2. Bank selection in a partially shared L2 cache for the base architecture with a sharing degree of 16. The 16 cores within each of the four clusters share their LLC banks among them. The numbers on the tiles correspond to their identifiers. We show the bit correspondence used by the cores in the marked cluster that given a bank selector generates the tile identifier whose LLC bank stores the block.

bits in the cache bank identifier. This correspondence works in such a way that the cores within the cluster access the banks within the cluster.

2.1. Coherence Issues

In a partially shared LLC the cores within each cluster share their L2 banks, an therefore two levels of coherence are needed: a first level to keep coherence among the private L1 caches of the tiles that share L2 banks, and a second level to keep coherence between the L2 banks shared by different clusters of tiles. We assume a directory in the L2 caches for the first level of coherence and a directory cache at the memory controller for the second level of coherence. Eight memory controllers are placed along the edges of the chip and memory addresses are interleaved across them.

In order to keep cache coherence, every time that some data is transferred from a shared level to a lower private level, the shared level needs to track the private copies of the block. The total overhead depends on the size of the caches whose contents must be tracked, and on their sharing degree. For instance, L2 caches are several times larger than L1 caches, hence the directory needed for tracking the copies of blocks in private L2 caches will be several times bigger than the one needed for tracking the copies in private L1 caches.

When it comes to partially shared LLCs, each LLC bank is shared by a number of private caches, and blocks in private caches must be tracked. At the same time, since the LLC is partially shared, the blocks stored in the LLC banks are shared within the same LLC cluster but private with respect to another LLC cluster. Therefore, these blocks must also be tracked. This creates the need to have two levels of coherence to support partially shared LLCs, with sharing information in both levels. However, having two levels of coherence information does not mean an increase in the total directory overhead compared to a completely private or a completely shared LLC, which only requires one level. On the contrary, the overhead can be reduced by using two levels. For instance, lets assume a 64-core tiled CMP with L1 caches, L2 caches and main memory. If the L2 cache is shared among all cores, 64-bit vectors are needed to track the copies of the blocks in the L1 caches. If the L2 banks are private to the cores, 64-bit vectors are needed again, this time to track the copies in the L2 banks. However, if a partially shared L2 with a sharing degree of 8 is used, 8-bit vectors are needed to track all the possible copies in each cluster of eight L1 caches, and 8-bit vectors are needed to track all the possible copies in the 64/8 L2 banks that can hold private copies

Table I. Sharing Information Overhead of Partially Shared Caches on the Total Cache Capacity of the Chip

Cores	Sharing Degree									
	1	2	4	8	16	32	64	128	256	512
64	44%	22%	11%	6.3%	4.2%	4.2%	5.6%			
128	89%	45%	23%	12%	7%	5.6%	6.9%	11%		
256	178%	89%	45%	23%	13%	8.3%	8.3%	13%	22%	
512	356%	178%	89%	45%	24%	14%	11%	14%	24%	44%

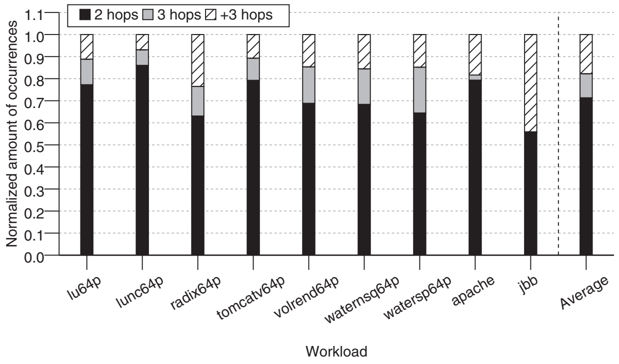


Fig. 3. Cache miss breakdown.

of the same blocks for different L1 clusters. Smaller entries result in a smaller total overhead.

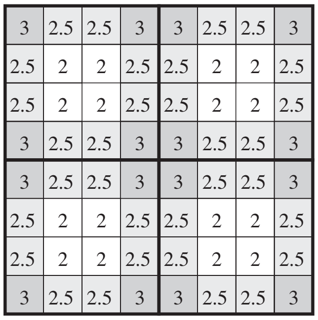
Table I shows the overhead introduced to store the sharing vectors for several chips with different numbers of cores and sharing degrees, assuming in all cases a 4x over-provisioned sparse directory and L2 banks that are eight times bigger than the L1 banks. The smallest overhead is found in an intermediate partially shared organization.

2.2. Block Mapping and Distance to Access the LLC

The LLC acts as a last barrier to prevent costly requests to main memory. In order to attain good performance, the miss ratio of the LLC must be very low. Hence, to be able to achieve good performance in a two level cache hierarchy like the one assumed in this paper, most L1 cache misses should be resolved by retrieving the desired data from the L2 cache. Figure 3 shows the breakdown of L1 cache misses in three categories for a number of workloads executed in the machine described in the evaluation section (see Table III). We consider that a hop is the sending of a message from a certain origin to a certain destination. Depending on the number of hops involved, we differentiate three kinds of L1 cache misses.

2-hop misses. A request message for the accessed block is sent from the L1 cache to the home L2 cache bank determined by the block mapping policy (first hop). The block is found in that L2 bank, which sends it in a response message (second hop).

3-hop misses. A request message for the accessed block is sent from the L1 cache to the home L2 cache bank determined by the block mapping policy (first hop). There is no valid copy of the block in the L2 bank, but directory information is found indicating that there is a valid copy in an L1 cache. The request message is forwarded to that L1 cache (second hop). Upon the reception of the forwarded request, the L1 cache sends a message containing a valid copy of the block to the requestor (third hop).



3	2.5	2.5	3	3	2.5	2.5	3
2.5	2	2	2.5	2.5	2	2	2.5
2.5	2	2	2.5	2.5	2	2	2.5
3	2.5	2.5	3	3	2.5	2.5	3
3	2.5	2.5	3	3	2.5	2.5	3
2.5	2	2	2.5	2.5	2	2	2.5
2.5	2	2	2.5	2.5	2	2	2.5
3	2.5	2.5	3	3	2.5	2.5	3

Fig. 4. Average distance in number of links from the core to the LLC banks that it accesses in a 64-tile CMP with a sharing degree of 16 (four clusters).

+3-hop misses. A request message for the accessed block is sent from the L1 cache to the home L2 cache bank determined by the block mapping policy. There is no valid copy of the block in the L2 bank, and the directory information indicates that there are no valid copies in the set of L1 caches that access that L2 bank. The request message is forwarded to the memory controller. This kind of miss requires at least four hops (in the case that the block is retrieved from memory) or even more hops if another L2 bank must be accessed to find a valid copy. For simplicity, we do not further divide +3-hop misses in different types in our analysis.

The main conclusion drawn from this figure is that most L1 misses belong to the 2-hop miss category, in which the traversal of links to and from the LLC accounts for most of the latency to solve the misses. No single benchmark shows a smaller amount than 50% of 2-hop misses, and on average over 70% of misses are 2-hop misses. Jbb is the benchmark with the smaller amount of 2-hop misses (55%) and it is due to the huge L2 miss rate that this workload incurs, over 40%, due to its enormous working set. 3-hop misses represent also a significant fraction of the total number of L1 cache misses of some workloads, and these misses are significantly affected by the distance to the home LLC bank too.

Therefore, the average L1 cache miss latency is mainly determined by the distance from the requesting core to the L2 bank in which the data is located. As a consequence, the mapping of blocks to cache banks is key for improving the performance of the cache hierarchy of a CMP, since it determines the distance from a core to the LLC banks that it accesses.

Traditional partially shared caches do nothing to reduce the distance from the cores to the LLC banks that they access. They assume that each cluster of cores share their LLC banks, which results in cache accesses with high latency, especially for those cores that are located far from the center of the cluster. To illustrate this, Figure 4 shows the average distance in number of links from each core to the LLC banks it accesses in the traditional partially shared cache organization that we take as a baseline in this paper, for 64 cores and a sharing degree of 16.

We can see in Figure 4 that, as we move towards the edges of the clusters, the average distance from a core to the LLC banks of the cluster grows. For higher number of cores and higher sharing degrees, the differences among tiles will be even larger. This kind of partially shared cache organization made sense when clusters of cores were located in different chips because accessing the LLC banks within the chip is faster than accessing banks in another chip. However, with many-cores in which all the cores are located in the same die, a partially shared cache organization like this do not optimize the access to the LLC banks.

3. DISTANCE-AWARE PARTIALLY SHARED CACHE ORGANIZATIONS

A shared LLC organization is defined by two elements. The first one is the access relationship between cores and LLC banks: each core accesses s LLC banks, where s is the sharing degree. The other one is the labeling of LLC banks. Each LLC bank stores data corresponding to a portion of the memory space, which is determined by its *bank label*. Then, we must make sure that every core accesses one LLC bank for each bank label so that the core can access data from the whole memory space.

By allowing all cores to access different subsets of the LLC banks we can make each core access its closest banks and therefore optimize the distance from each core to its LLC banks. To achieve this, the mapping function that is used for finding the home LLC bank of each block will be different for each core.

Formally stated, a partially shared LLC organization is represented by a labeled balanced directed graph where each vertex represents a tile in the CMP and each arc represents a core (tail of the arc) accessing an LLC bank (head of the arc). Each arc has a weight that corresponds to the number of links traversed to go from the source tile to the destination tile. The weight of an arc depends on the placement of the core and the LLC and on the underlying network topology that connects the tiles. The directed graph is balanced because all the cores access the same number of LLC banks, and all LLC banks are accessed by the same number of cores. Therefore, the indegrees and outdegrees of every vertex are equal to the sharing degree, s, which must be a divisor of the number of vertices, $|V|$. There are $|V|/s$ vertices labeled with every number between 0 and $s - 1$, representing which portion of the memory space is mapped to the LLC bank of the tile. These numbers are the bank labels of the LLC banks. In addition, two arcs with the same tail cannot have heads with the same bank label, because each core needs to access one and only one LLC bank for each possible bank label.

In order to find the best possible partially shared LLC organization we must search for a graph that has the properties stated above and, at the same time, minimizes the total sum of the weights of its arcs (i.e. the distance from the cores to the LLC banks). The resulting cache organization will improve the performance of the system and will reduce the power consumption of the network. We call these configurations of the chip "Distance-Aware Partially Shared Cache Organizations" (DAPSCO). DAPSCO does not increase the complexity of the coherence protocol nor the coherence information overhead compared to traditional partially shared cache organizations, as we show in Section 3.3.

Finding the optimal DAPSCO for a given CMP size and sharing degree is an NP-complete problem, and we explain how to use heuristic algorithms to find near optimal configurations in Section 3.1. However, finding the optimal DAPSCO is easier for CMPs with torus based interconnection networks due to their symmetry, and a method for doing so in which every core uses the same pattern to access the LLC will be explained in Section 3.2.

3.1. Exploring the DAPSCO Search Space

In order to find a DAPSCO as good as possible for both meshes and tori, we have used two well-known global optimization algorithms: a hill climbing algorithm and a simulated annealing algorithm. The methodology explained in this section is applicable to any interconnection network topology. These heuristic algorithms start with the traditional partially shared LLC organization in which clusters of cores share their LLC banks. This is a valid organization since it meets all the constraints of the graph problem stated before. Then, the algorithms evaluate random valid organizations that originate by applying two different operators.

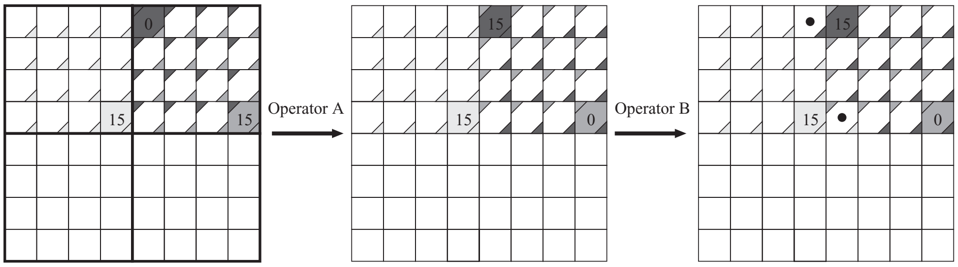


Fig. 5. Application of a sequence of operators to the initial organization of the search to obtain an improved solution with a sharing degree of 16. The numbers represent LLC bank labels (0 and 15 for this example). The color of the upper left corner triangle of a core matches the color of the LLC bank accessed for label 0. The lower right corner triangle does the same for label 15. The central figure shows the result of applying Operator A to two LLC banks, which interchange their labels (0 and 15). Any core that accessed one of these LLC banks for label 0 has to access the other LLC bank after the application (notice the change of colors in the triangles). The figure on the right shows the result of applying Operator B to the dotted cores, which interchange their accessed LLC banks for bank label 15 (again, notice the change of colors in the triangles of those tiles). This results in a distance reduction of 2 links between each of these cores and their accessed LLC banks for bank label 15, which are now adjacent to the cores.

—*Operator* A. Two LLC banks that have different bank labels interchange their bank labels. This also implies that all the cores that previously accessed one of these LLC banks for a particular bank label now have to access the other LLC bank.

—*Operator* B. Two cores that access two different LLC banks for the same bank label interchange their LLC banks so that they access each other's previous LLC bank.

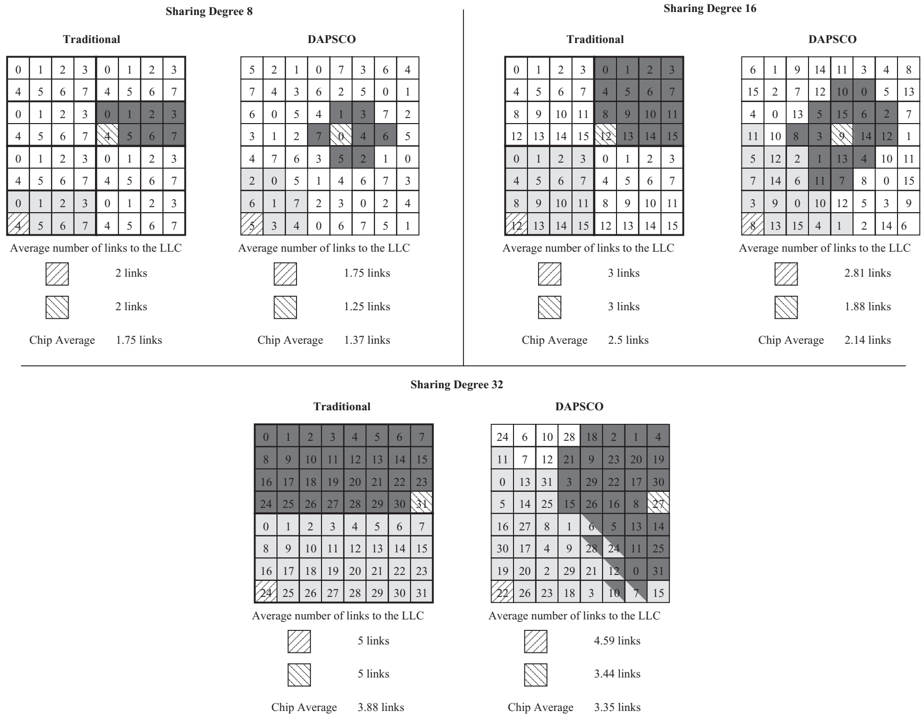
It is straightforward that any LLC organization resulting from applying these two operators to a valid organization is also valid. Furthermore, every valid organization can be reached from any other valid organization by applying an appropriate sequence of these two operators. Figure 5 shows the application of the two operators on the initial cache organization, providing immediate improvements in the distance from the cores to the LLC.

Figure 6 compares the LLC access patterns of some cores in the traditional partially shared LLC organization (the one used as the initial state of our search algorithms) and in the best DAPSCO found by our algorithms, for a 64-core CMP with sharing degrees of 8, 16 and 32 and a mesh network.

3.2. Tori and Sliding Patterns

In the case of tori, a simpler method can be used to obtain an optimal (or near optimal) partially shared cache organization. We call "access pattern" to the shape of the group of tiles accessed by a core. Thanks to the symmetry of tori, the same access pattern can be used by every core to access the LCC banks if that access pattern can be used to tessellate the chip, as explained below in this section. We call these access patterns "sliding patterns".

In order to find an optimal sliding pattern we should start by finding all the access patterns with minimum total distance. For this, we start with choosing any tile as the central tile for the pattern. This central tile represents both the core that uses the pattern and one of the LLC banks accessed by the core. Then, the nearest tile to the center is added, representing another LLC bank accessed by the core in the central tile. Tiles are added following this policy until the pattern contains s tiles. Since several tiles may be at the same distance from the central tile, any one of them can be added at each step resulting in different candidate access patterns. It is straightforward that



Sharing Degree 8, Traditional

0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7

Sharing Degree 8, DAPSCO

5	2	1	0	7	3	6	4
7	4	3	6	2	5	0	1
6	0	5	4	1	3	7	2
3	1	2	7	0	4	6	5
4	7	6	3	5	2	1	0
2	0	5	1	4	6	7	3
6	1	7	2	3	0	2	4
5	3	4	0	6	7	5	1

Sharing Degree 16, Traditional

0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15

Sharing Degree 16, DAPSCO

6	1	9	14	11	3	4	8
15	2	7	12	10	0	5	13
4	0	13	5	15	6	2	7
11	10	8	3	9	14	12	1
5	12	2	1	13	4	10	11
7	14	6	11	7	8	0	15
3	9	0	10	12	5	3	9
8	13	15	4	1	2	14	6

Sharing Degree 32, Traditional

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Sharing Degree 32, DAPSCO

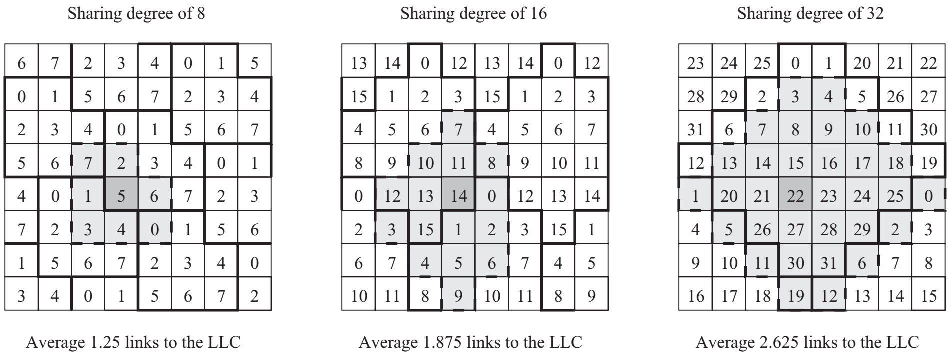
24	6	10	28	18	2	1	4
11	7	12	21	9	23	20	19
0	13	31	3	29	22	17	30
5	14	25	15	26	16	8	27
16	27	8	1	6	5	13	14
30	17	4	9	28	24	11	25
19	20	2	29	21	12	0	31
22	26	23	18	3	10	7	15

Fig. 6. LLC banks accessed by the cores in a mesh. Sharing degrees of 8, 16 and 32. Traditional partially shared LLC organization (left) and DAPSCO (right). Each striped core accesses the shaded LLC banks surrounding it. The numbers on the tiles represent the bank label. Notice that each core always accesses one LLC bank for each bank label. The average number of links traversed to reach the LLC banks is shown under the figures. DAPSCO significantly reduces this number of links.

the distance between the core and the LLC banks in all the resulting patterns is the minimum possible.

As we said before, a cache organization must ensure that every core accesses one LLC bank for each bank label so that the core has access to the whole memory space. Hence, we must assign bank labels to the tiles in such a way that the access pattern allows every core to access one LLC bank for each label. Unfortunately, this can be done only for patterns which can tessellate the chip (sliding patterns). In our case, a tessellation consists of dividing the chip in several polygons with the shape of the pattern. These polygons must not overlap and must cover all of the tiles of the chip. When tessellating, the symmetric topology of the torus network must be taken into account and a polygon that spans beyond one edge of the chip continues in the tessellation by the other end of the chip (that is, it wraps around). All the polygons must have the same orientation (that is, they cannot be flipped or rotated). If such a tessellation exists, the access pattern is a sliding pattern. The bank labels can be assigned to the tiles in any order as long as it is the same order for every polygon in the tessellation. This ensures that every core can access one LLC bank for each label by means of the sliding pattern.

In Figure 7 we show the optimal sliding patterns and label assignations in a 64-tile CMP with a torus interconnection network for sharing degrees of 8, 16 and 32. Notice that no tessellation of a 64-tile chip exists with a minimum distance pattern for sharing degree of 16 (we explored the full search space to check it), and a sub-optimal sliding pattern with an additional link in the total distance is shown instead. When no optimal



Sharing degree of 8

6	7	2	3	4	0	1	5
0	1	5	6	7	2	3	4
2	3	4	0	1	5	6	7
5	6	7	2	3	4	0	1
4	0	1	5	6	7	2	3
7	2	3	4	0	1	5	6
1	5	6	7	2	3	4	0
3	4	0	1	5	6	7	2

Average 1.25 links to the LLC

Sharing degree of 16

13	14	0	12	13	14	0	12
15	1	2	3	15	1	2	3
4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11
0	12	13	14	0	12	13	14
2	3	15	1	2	3	15	1
6	7	4	5	6	7	4	5
10	11	8	9	10	11	8	9

Average 1.875 links to the LLC

Sharing degree of 32

23	24	25	0	1	20	21	22
28	29	2	3	4	5	26	27
31	6	7	8	9	10	11	30
12	13	14	15	16	17	18	19
1	20	21	22	23	24	25	0
4	5	26	27	28	29	2	3
9	10	11	30	31	6	7	8
16	17	18	19	12	13	14	15

Average 2.625 links to the LLC

Fig. 7. Tessellations (solid lines) and label assignations to LLC banks for sharing degrees of 8, 16 and 32 in a 64-tile CMP. The dashed lines and the shaded tiles show the sliding pattern applied to the dark gray tile. The resulting average number of links to access the LLC in each configuration is shown. Compare these values with those of a traditional partially shared LLC with a torus: 1.75 links for a sharing degree of 8, 2.5 for a sharing degree of 16 and 3.25 for a sharing degree of 32.

sliding pattern exists, the global optimization algorithms explained in section 3.1 may find a solution better than the best possible sliding pattern. This is the case of a 64-tile CMP with a sharing degree of 16.

3.3. Implementation Details

Only small changes to the hardware of traditional partially shared cache organizations are needed to implement DAPSCO. Assuming a two level cache hierarchy, three families of mapping functions used by the CMP must be modified. Notice that these functions already exist in the circuitry of a CMP with a partially shared LLC, and DAPSCO only needs different ones.

The first family of functions (family 1 from now on) maps block addresses to L2 banks. Each core needs one function that, given the bank selector of the address of the block, chooses the predetermined L2 bank that is always accessed by that particular core for that bank selector upon an L1 cache miss or a write-back.

The second family of functions (family 2 from now on) maps bits of the sharing vector of the first level directory to identifiers of L1 banks. Each L2 bank needs one function to map each of the s bits of the sharing vector (s is the sharing degree) to one of the s L1 banks that can hold copies of the block provided by that L2 bank.

The third family of functions (family 3 from now on) maps bits of the sharing vector of the second level directory to identifiers of L2 banks. Given the bank selector contained in the address of a block, there are n/s LLC banks with the corresponding bank label (n is the number of tiles in the CMP) which may contain a copy of the block. s different functions are necessary to perform this task, one per bank label.

These three families of mapping functions can be implemented in a number of ways. We have considered two options: using small tables or using small combinational cicuits.

The simplest and most flexible way of implementing these functions is by using one small table for each one. For family 1, the size of each table would be $s \times log_2(n)$ bits. Each entry of the table represents the LLC bank that must be accessed for a given bank label. There is one table per core in the chip.

For each function of family 2, a table of size $s \times log_2(n)$ bits is needed. Each entry represents the L1 cache corresponding to a given bit in the sharing vector. There is one table per L2 bank in the chip.

Table II. Sizes of DAPSCO's Address-to-Bank Circuits for a Sharing Degree of 8

Number of cores	Average number of transistors per tile	No. of transistors in the longest critical path of any circuit
64	64	8
128	72	8
256	72	8
512	75	8

For each function of family 3, a table of size $n/s \times log_2(n)$ bits is needed. There are s of these tables, and they are accessed as follows: given a block address, the bank label corresponding to the block is used to choose the table to be accessed; given a bit of a sharing vector of the second level directory, the corresponding entry of the chosen table is looked up to get the LLC bank represented by the bit.

The contents of these tables would be different in DAPSCO in order to use the new optimized mapping. Both the traditional partially shared organization and DAPSCO would need exactly the same hardware with this implementation approach.

Alternatively, these tables can be optimized to simple combinational circuits so as to reduce their overhead, in the case of both the traditional mapping and DAPSCO.

In order to calculate the overhead of DAPSCO using this approach, we have written code that generates the layout of the three families of circuits. We have calculated these layouts for the baseline partially shared LLC and also for DAPSCO, in order to compare both organizations. We have found that, in the case of the circuits that translate sharing code into L1 bank identifiers (i.e. families 2 and 3), there are no noticeable differences between fixed-boundary clusters and DAPSCO in neither the total number of transistors nor the number of transistors in the critical path of these circuits. Therefore, no overhead is introduced by DAPSCO due to its different implementation of the circuits of families 2 and 3.

However, the address-to-bank mapping circuits (family 1), is indeed more complex in DAPSCO. This address-to-bank mapping is straightforward in the baseline partially shared LLC, since the label bits of the block address can be used to generate the identifier of the L2 bank to access (maybe with some reordering of the bits), while DAPSCO needs a circuit to perform this task. Table II shows the number of transistors needed by the address-to-bank map circuits of DAPSCO when a mesh and a sharing degree of 8 are used. This table shows the average number of transistors per tile and the maximum length of the critical path of any of these circuits in the chip. Fortunately, these additional transistors are negligible in terms of power and area. In addition, the overhead of this circuit scales gracefully with the number of cores, and the maximum critical path remains constant for any number of cores tested. As for latency, this circuit is easily traversed in one cycle and it is accessed in parallel to the L1 tags. Therefore, no latency is added to the critical path of a cache miss. As a conclusion, no noticeable increase in power, area or latency is produced by any of the modifications of the circuitry needed by DAPSCO.

4. EVALUATION

4.1. Effectiveness of the DAPSCO Configurations Found

In order to search for the best possible DAPSCO, we use both a hill-climbing and a simulated annealing algorithm, as noted in Section 3.1. These algorithms were executed repeatedly for several combinations of sharing degrees and number of cores, and the best solution that they found for each combination was chosen. In total, the algorithms ran on machines equipped with 2.33GHz Intel Xeon processors for approximately one week. They were restarted when the consecutive generation of two million LLC organizations did not improve the best organization found by the current execution of

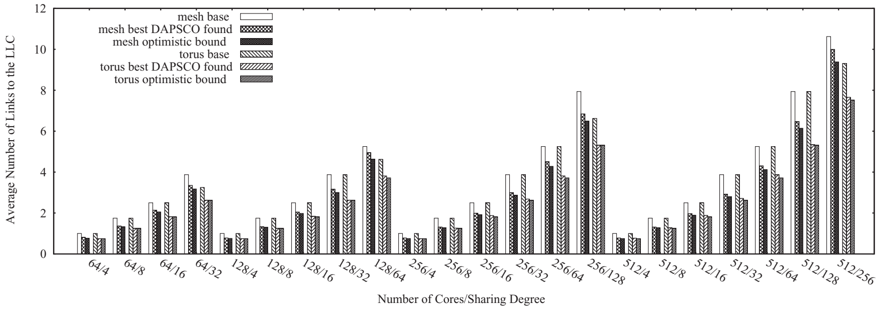


Fig. 8. Average number of links from a core to an arbitrary LLC bank. X-axis labels represent the number of cores in the CMP (from 64 to 512) and the sharing degree (from 4 to 256).

the algorithm. We tested some optimizations such as only applying operators that improved the average distance to the LLC from one of the cores affected by the operators in order to direct the search.

Figure 8 shows the average number of links traversed for the cores to access the LLC using the best configurations found by the algorithms. These results are shown for meshes and tori. Three different values can be observed for each topology: that of the base mapping, that of an optimistic bound and that of the best DAPSCO found by our algorithms. The optimistic bound assumes that every core accesses its closest LLC banks. This is not always possible in practice, but we include it as an upper bound to check the effectiveness of the heuristics employed. In fact, no real DAPSCO configuration can match the optimistic DAPSCO in a mesh for sharing degrees of four or more. Intuitively, this is because if every core accessed its closest LLC banks, the banks in the corners (and borders) of the chip would be accessed by less cores than those banks in the center of the chip, and the configuration would not be valid. As the sharing degree increases, this problem becomes more noticeable.

One important fact is that as the sharing degree increases, the number of links to access the LLC grows too in all configurations, although the relative improvement achieved by DAPSCO remains similar. Hence, our proposal will have more beneficial effects in the performance of the system for higher sharing degrees, because in those cases fixed latencies (such as the accesses to the tag and data arrays of the caches) account for a smaller fraction of the total time to solve each cache miss while the latency to traverse links and routers takes a larger fraction.

4.2. Simulation Methodology

We use the GEMS [Martin et al. 2005] simulator to model a tiled-CMP whose characteristics can be seen in Table III. We use the workloads described in Table IV. For virtualized workloads we use Virtual-GEMS [García-Guirado et al. 2009]. We model a detailed interconnection network with the Garnet [Agarwal et al. 2009] network simulator. Energy consumption figures for the network are obtained from the Orion 2.0 [Kahng et al. 2009] power model.

Our goal is to compare the traditional partially shared cache organization described in Section 2 and DAPSCO. We have limited our evaluation to 64-tile CMPs due to simulation time constraints, despite the fact that DAPSCO would obtain more favorable results with more tiles. In total, we have evaluated four different machine configurations.

—8Traditional: traditional mapping for partially shared caches in which each cluster of 8 tiles share their L2 banks.
—8DAPSCO: DAPSCO with a sharing degree equal to 8.

Table III. System Characteristics

Processors	64 UltraSPARC-III+ 3 GHz. 2-ways, in-order.
L1 Cache	Split I&D. Size: 64KB. Associativity: 4-ways. 64 bytes/block. Access latency: 1 cycle.
L2 Cache	Size: 1MB each bank. 64MB total. Associativity: 8-ways. 64 bytes/block. Access latency: 2 (tag) + 3 (data) cycles.
RAM	4 GB DRAM. 8 memory controllers along the edges of the chip. Memory latency: 150 cycles + on-chip delay. Page Size: 4 KB.
Interconnection	Bi-dimensional mesh and torus 8x8. 16 byte links. Latency mesh: 2 cycles/link. Latency torus: 4 cycles/link. (in absence of contention) Flit Size: 16 bytes. Control packet size: 1 flit. Data packet size: 6 flits.

Table IV. Benchmark Configurations

Workload	Description	Size
lu64p	Factorization of a dense matrix	512 × 512 matrix
lunc64p	Factorization of a dense matrix, non-contiguous memory	512 × 512 matrix
tomcatv64p	Vectorized mesh generation	256
volrend64p	Ray-casting rendering	Head
watersp64p	Optimized molecular dynamic simulation of water	512 molecules
apache4x16p	Virtualized web server with static contents	Four 16-core VMs with 500 clients each, 10ms between requests of a client
jbb4x16p	Virtualized Java server	Four 16-core VMs with 1.5 warehouses per core

—16Traditional: traditional mapping for partially shared caches in which each cluster of 16 tiles share their L2 banks.
—16DAPSCO: DAPSCO with a sharing degree equal to 16.

Two different network topologies, mesh and torus, have been used in the tests. Torus links are longer than mesh links, and therefore we set their latency to twice that of mesh links. The suffixes Mesh or Torus are added to the names of the configurations in order to identify the network topology used.

The average number of links traversed by a message from an arbitrary core to reach an arbitrary bank of the LLC is shown in Table V. The rest of the machine remains the same for all the configurations (Table III).

4.3. Results and Discussion

Most L1 cache misses are solved with just two hops to retrieve the data from the L2 cache, as we discussed in Section 2.2. These misses benefit the most from DAPSCO in terms of traversed links since both hops get advantage from the reduced distance to the L2 bank accessed. Figures 9 and 10 show the average number of links traversed to solve these misses in the mesh and torus networks, respectively. These experimental results for 2-hop misses with the torus (improvements of 40% and 33% for sharing

Table V. Average Distance from Cores to LLC Banks

Configuration	Average distance to the LLC (links)	Improvement over traditional
8TraditionalMesh	1.75	
8DAPSCOMesh	1.37	24%
16TraditionalMesh	2.5	
16DAPSCOMesh	2.14	17%
8TraditionalTorus	1.75	
8DAPSCOTorus	1.25	40%
16TraditionalTorus	2.5	
16DAPSCOTorus	1.88	33%

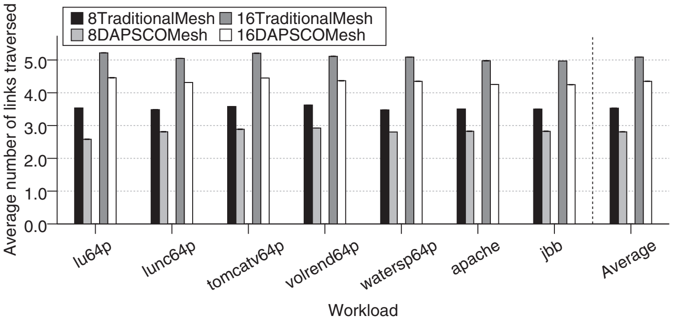


Fig. 9. Average number of links traversed to solve 2-hop cache misses using a mesh network.

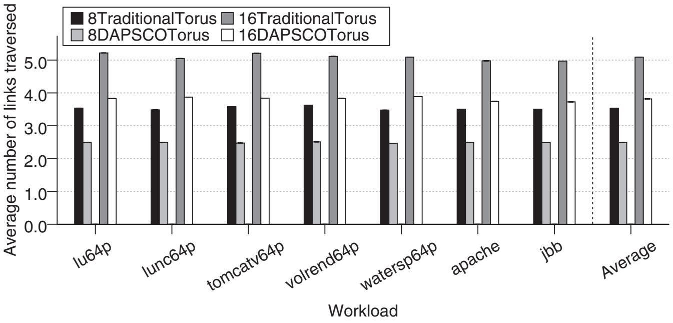


Fig. 10. Average number of links traversed to solve 2-hop cache misses using a torus network.

degrees of 8 and 16) and the mesh (improvements of 24% and 17% for sharing degrees of 8 and 16) match almost perfectly the values previously calculated for the average number of links traversed to access the LLC that we showed in Table V.

Other kinds of misses also experience a reduction in the number of traversed interconnection network links, although smaller. Since the 2-hop miss kind is the most frequent one and it gets full advantage of DAPSCO in all the hops of its critical path, it causes a noticeable impact on the overall latency and power consumption of cache misses.

In the end, the reduction in links traversed to solve misses translates directly into reductions in both execution time and network energy consumption. Figures 11 and 12 show the execution time of the eight configurations tested. In the case of the mesh (Figure 11), the performance of the system improves by 4% and 6% when using DAPSCO with sharing degrees of 8 and 16, respectively. In the case of the torus (Figure 12), the performance improvement of DAPSCO rises to 10% and 13% for

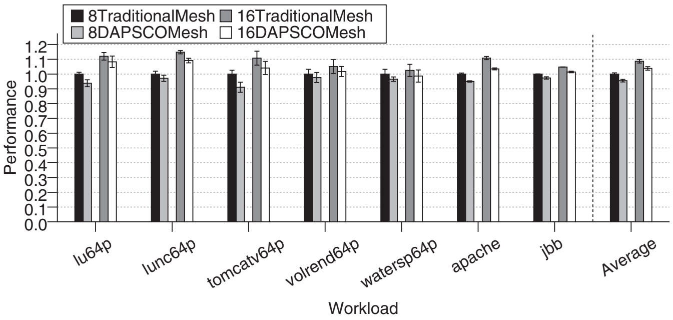


Fig. 11. Normalized execution time using a mesh interconnection network.

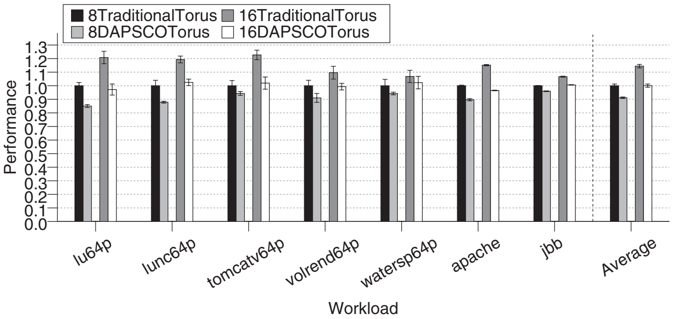


Fig. 12. Normalized execution time using a torus interconnection network.

sharing degrees of 8 and 16. We can see that despite the average reduction in traversed links being smaller in the case of a sharing degree of 16, the gain in execution time is slightly higher than for a sharing degree of 8. The main cause was mentioned previously: the distance traversed to solve cache misses is higher for higher sharing degrees, hence the percentage of cache miss latency due to link and router traversals increases in comparison with fixed latencies (e.g. cache tag and data array accesses), and this makes the benefits of DAPSCO more noticeable.

Figures 13 and 14 show the normalized energy consumption of the network for the torus and the mesh, respectively. Again, thanks to the reduction in links and routers traversed to solve misses, energy consumption gets reduced when using DAPSCO by 4% and 6% with a mesh and by 10% and 13% with a torus for sharing degrees of 8 and 16, respectively. There is a clear parallelism between the results of performance and network energy for every benchmark.

All of these results show the effectiveness of DAPSCO. In fact, in the worst benchmark for DAPSCO, the performance of the system gets improved by 2% with a mesh and by 5% with a torus.

However, the full potential of DAPSCO is not reached with the tested configurations. As the number of cores and the sharing degree of the partially shared caches grow, the benefits of DAPSCO will become more noticeable since both the percentage of links removed from the path to the LLC banks and the percentage of the execution time due to link traversals increase.

As of the performance comparison between sharing degrees, which is not the goal of this paper, in our tests a sharing degree of 8 performs better than a sharing degree of 16, but we believe that this could change with a wider set of workloads with larger working sets, as proven by other studies.

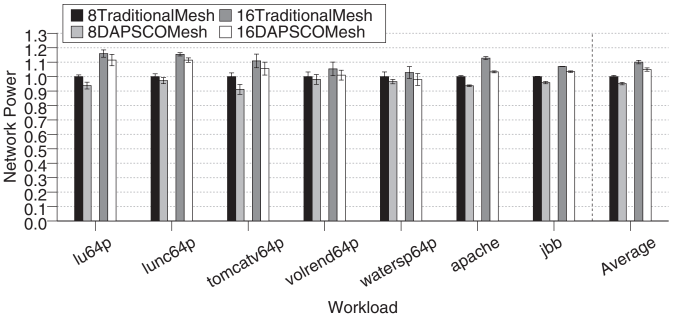


Fig. 13. Normalized energy consumption of the interconnection network using a mesh network.

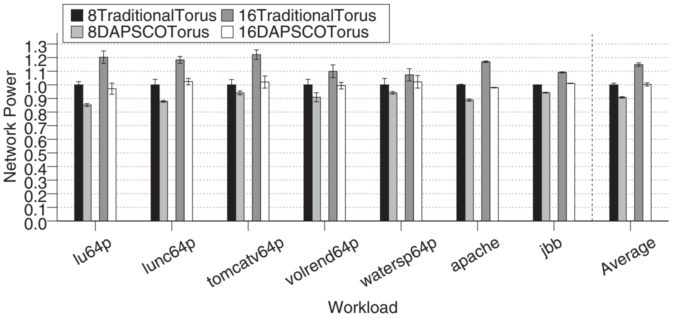


Fig. 14. Normalized energy consumption of the interconnection network using a torus network.

5. ORTHOGONALITY OF DAPSCO: REACTIVE NUCA

DAPSCO is orthogonal to other proposals that use traditional partially shared caches. In this section we show this by applying DAPSCO to one of these proposals known as Reactive NUCA [Hardavellas et al. 2009]. Reactive NUCA is based on dynamically clasifying the blocks accessed by the cores in several pre-determined types, and applying the best pre-determined sharing degree to each of these types to improve performance. Three different types were considered to which the blocks can belong: instructions, private data and shared data. It was determined in Hardavellas et al. [2009] that the sharing degrees that optimized performance for these types in a 16-core CMP with a torus network were: four for instructions, one for private data (equivalent to using private caches for private data) and sixteen for shared data (equivalent to using a shared cache that avoids block replication of shared data).

We have applied Reactive NUCA to a 64-core CMP with a mesh network. When a mesh is used, Reactive NUCA uses traditional partially shared caches for the LLCs. We have considered the same block types as the original proposal of Reactive NUCA (instructions, private data and shared data), and by performing exhaustive tests we have determined that the sharing degrees for these types of data that yield the best performance in our benchmarks are four for instructions, one for private data and eight for shared data. Then, we have modified Reactive NUCA to replace the mapping of the partially shared caches with DAPSCO. Figure 15 shows the performance comparison between Reactive NUCA and Reactive NUCA plus DAPSCO. These results show that the performance of Reactive NUCA improves by 3% in average (and up to 7% in some benchmarks) when DAPSCO is enabled, thanks to the reduction in the links traversed to access the LLC banks provided by DAPSCO.

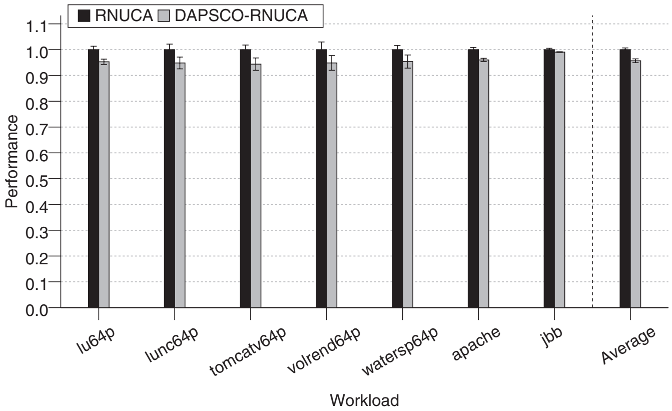


Fig. 15. Performance of Reactive NUCA without and with DAPSCO.

6. RELATED WORK

Several works study the effect of using partially shared caches of different sizes. Huh et al. [2005] test five different sharing degrees for a 16-core CMP, from a completely private LLCs to a completely shared LLC, by using a traditional block mapping for partially shared caches. Their results show that the sharing degree that achieves the best performance depends on the workload executed.

Hammoud et al. [2009] show that different sharing degrees of the LLC are more appropriate for different execution phases of the same application. They develop a technique for dynamically varying the sharing degree of the LLC based on real-time feedback on the behavior of the applications. DAPSCO could be used along with this technique instead of the traditional block mapping to improve the performance and energy consumption of the system.

Dynamic (or adaptive) mappings provide the flexibility that the cache bank where a block is mapped is not fixed, which can be used to improve performance by dynamically bringing blocks to banks closer to the requesting cores. However, complex and power consuming lookup mechanisms and extra structures are needed to locate blocks in this case and the cache coherence protocol may become more complex. On the other hand, static mappings like DAPSCO map each memory block to a fixed cache bank, are simpler to implement than dynamic mappings and have been traditionally used in commercial machines [Kongetira et al. 2005; Shah et al. 2007].

Regarding cache miss latency in CMPs, many proposals [Kim et al. 2002; Chang and Sohi 2006; Beckmann et al. 2006; Hardavellas et al. 2009] reduce it by dynamically trying to allocate copies of the block as close as possible to the requestors, but these techniques commonly increase network traffic and need power-consuming lookup mechanisms to locate blocks and extra structures. In contrast, DAPSCO consumes less energy as a result of the miss latency reduction and the reduction in links traversed by each message and requires almost no extra hardware.

Reactive NUCA [Hardavellas et al. 2009] introduces the concept of fixed-center clusters which, similarly to the sliding patterns described as a part of DAPSCO, make each core access a different subset of the L2 banks to replicate data without increasing the capacity pressure of the cache and enable fast nearest-neighbor communication. However, fixed-center clusters only work for torus networks, which can be seen as an important shortcoming since current CMPs proposals commonly use mesh networks which are easier to implement and cheaper. Additionally, the rotational interleaving that is used to create these clusters produces sub-optimal distances to the LLC, and cluster size must be a power of two and smaller than half the number of cache banks of the CMP. For instance, rotational interleaving does not work for 32-bank clusters in a 64-core CMP. In fact, DAPSCO can be adapted to Reactive NUCA to further improve its energy consumption and performance in large CMPs, and also to improve the use of meshes with Reactive-NUCA, as shown in Section 5.

The operating system can take into account the address-to-cache-bank mapping when performing the virtual to physical address translation in order to map memory pages to certain cache banks so as to improve the performance of the system [Cho and Jin 2006]. Distance-Aware Round-Robin Mapping [Ros et al. 2009] improves the mapping of the memory pages to cache banks in large NUCA caches by means of an OS-managed mechanism that both reduces the distance from the data to the requestors and provides a fair utilization of the NUCA banks.

Plenty of research on reducing energy consumption in CMPs is being carried out. Regarding NoCs, cache coherence protocols can take advantage of heterogeneous networks to reduce power consumption by transmitting critical, short messages through fast power-consuming wires and non-critical messages through slower low-power wires [Flores et al. 2010]. As for cache architecture, TurboTag [Lotfi-Kamran et al. 2010] uses bloom filters to avoid unnecessary tag lookups and reduce power consumption. However, all these proposals reduce energy consumption at the cost of degrading performance, while DAPSCO improves it.

Finally, a new tag directory buffer technique has been recently proposed as a good energy-delay compromise between private and shared cache organizations [Hughes et al. 2010].

7. CONCLUSIONS

We have proposed DAPSCO to optimize the cache organization of tiled CMPs by making each core access the LLC banks surrounding it in both mesh and torus topologies, minimizing the average number of interconnection network links traversed to access the LLC. Two heuristic algorithms have been used to explore the search space of the NP-complete problem of finding optimal DAPSCO mappings for CMPs using any topology, and a method for constructing optimal sliding patterns for torus-based CMPs has been presented.

The costs of DAPSCO are negligible in terms of hardware, and it achieves significant improvements in both the execution time of the system and the energy consumption of the interconnection network when compared to the traditional partially shared cache organization in which clusters of the tiles share their LLC banks.

We have shown two examples of DAPSCO that improve the performance of a 64-core CMP by 4% and 6% with an underlying mesh topology and by 10% and 13% with an underlying torus topology, all of it with respect to traditional partially shared caches with sharing degrees of 8 and 16. Network power consumption also gets reduced by 4% and 6% (mesh), and 10% and 13% (torus) regarding the same traditional configurations. We have also shown that, as the number of cores increases, DAPSCO removes a higher percentage of links from the critical path of cache misses, and the impact of link traversals account for a higher percentage of the execution time, making DAPSCO even more effective.

REFERENCES

AMD. 2007. AMD Athlon 64 X2 Dual-core processor product data sheet.

Agarwal, N., Krishna, T., Peh, L.-S., and Jha, N. K. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 33–42.

AGNY, R., DELANO, E., KUMAR, M., NACHIMUTHU, M., AND SHIVELEY, R. 2010. The Intel Itanium Processor 9300 series. Intel white paper.

BECKMANN, B. M., MARTY, M. R., AND WOOD, D. A. 2006. ASR: Adaptive selective replication for CMP caches. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 443–454.

CHANG, J. AND SOHI, G. S. 2006. Cooperative caching for chip multiprocessors. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA)*. 264–276.

CHO, S. AND JIN, L. 2006. Managing distributed, shared L2 caches through OS-level page allocation. In *Proceedings of the IEEE/ACM International Symposium on Michroarchitecture (MICRO)*. 455–468.

FLORES, A., ARAGÓN, J. L., AND ACACIO, M. E. 2010. Heterogeneous interconnects for energy-efficient message management in CMPs. *IEEE Trans. Computers 59,* 1, 16–28.

GARCÍA-GUIRADO, A., FERNÁNDEZ-PASCUAL, R., AND GARCÍA, J. M. 2009. Virtual-GEMS: An infrastructure to simulate virtual machines. In *Proceedings of the 5th International Workshop on Modeling, Benchmarking and Simulation* (in conjunction with *ISCA*). 53–62.

HAMMOUD, M., CHO, S., AND MELHEM, R. 2009. Dynamic cache clustering for chip multiprocessors. In *Proceedings of the 23rd International Conference on Supercomputing (ICS)*. 56–67.

HARDAVELLAS, N., FERDMAN, M., FALSAFI, B., AND AILAMAKI, A. 2009. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*. 184–195.

HELD, J. AND KOEHL, S. 2006. Inside Intel core microarchitecture. Intel white paper.

HUGHES, C., KIM, C., AND CHEN, Y.-K. 2010. Performance and energy implications of many-core caches for throughput computing. *IEEE Micro 30,* 6, 25–35.

HUH, J., KIM, C., SHAFI, H., ZHANG, L., BURGER, D., AND KECKLER, S. W. 2005. A NUCA substrate for flexible CMP cache sharing. In *Proceedings of the 19th Annual International Conference on Supercomputing (ICS)*. 31–40.

KAHNG, A. B., LI, B., PEH, L.-S., AND SAMADI, K. 2009. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 423–428.

KALLA, R., SINHAROY, B., STARKE, W. J., AND FLOYD, M. 2010. Power7: IBM's next-generation server processor. *IEEE Micro 30,* 2, 7–15.

KIM, C., BURGER, D., AND KECKLER, S. W. 2002. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 211–222.

KONGETIRA, P., AINGARAN, K., AND OLUKOTUN, K. 2005. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro 25,* 2, 21–29.

LIU, C., SIVASUBRAMANIAM, A., AND KANDEMIR, M. 2004. Organizing the last line of defense before hitting the memory wall for CMPs. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture (HPCA)*. 176–185.

LOTFI-KAMRAN, P., FERDMAN, M., CRISAN, D., AND FALSAFI, B. 2010. TurboTag: Lookup filtering to reduce coherence directory power. In *Proceedings of the 16th International Symposium on Low Power Electronics and Design (ISLPED)*. 377–382.

MARTIN, M. M. K., SORIN, D. J., BECKMANN, B. M., MARTY, M. R., XU, M., ALAMELDEEN, A. R., MOORE, K. E., HILL, M. D., AND WOOD, D. A. 2005. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *SIGARCH Comput. Architec. News 33,* 4, 92–99.

NAYFEH, B. A., OLUKOTUN, K., AND SINGH, J. P. 1996. The impact of shared-cache clustering in small-scale shared-memory multiprocessors. In *Proceedings of the 2nd IEEE Symposium on High-Performance Computer Architecture (HPCA)*. 74–84.

ROGERS, B. M., KRISHNA, A., BELL, G. B., VU, K., JIANG, X., AND SOLIHIN, Y. 2009. Scaling the bandwidth wall: Challenges in and avenues for CMP scaling. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA)*. 371–382.

ROS, A., CINTRA, M., ACACIO, M. E., AND GARCÍA, J. M. 2009. Distance-aware round-robin mapping for large NUCA caches. In *Proceedings of the 16th International Conference on High Performance Computing (HiPC)*. 79–88.

SHAH, M., BARREN, J., BROOKS, J., GOLLA, R., GROHOSKI, G., GURA, N., HETHERINGTON, R., JORDAN, P., LUTTRELL, M., OLSON, C., SANA, B., SHEAHAN, D., SPRACKLEN, L., AND WYNN, A. 2007. UltraSPARC T2: A highly-threaded, power-efficient, SPARC SoC. In *Proceedings of the IEEE Asian Solid-State Circuits Conference*. 22–25.

Received July 2011; revised October 2011 and November 2011; accepted December 2011