Power-Efficient Spilling Techniques for Chip Multiprocessors

Enric Herrero[1], José González[2], and Ramon Canal[1]

[1] Dept. d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
{eherrero,rcanal}@ac.upc.edu
[2] Larrabee Architecture Group
Intel Barcelona
pepe.gonzalez@intel.com

Abstract. Current trends in CMPs indicate that the core count will increase in the near future. One of the main performance limiters of these forthcoming microarchitectures is the latency and high-demand of the on-chip network and the off-chip memory communication. To optimize the usage of on-chip memory space and reduce off-chip traffic several techniques have proposed to use the N-chance forwarding mechanism, a solution for distributing unused cache space in chip multiprocessors. This technique, however, can lead in some cases to extra unnecessary network traffic or inefficient cache allocation. This paper presents two alternative power-efficient spilling methods to improve the efficiency of the N-chance forwarding mechanism. Compared to traditional Spilling, our Distance-Aware Spilling technique provides an energy efficiency improvement (MIPS^3/W) of 16% on average, and a reduction of the network usage of 14% in a ring configuration while increasing performance 6%. Our Selective Spilling technique is able to avoid most of the unnecessary reallocations and it doubles the reuse of spilled blocks, reducing network traffic by an average of 22%. A combination of both techniques allows to reduce the network usage by 30% on average without degrading performance, allowing a 9% increase of the energy efficiency.

1 Introduction

In the era of chip multiprocessors, traditional performance limiters such as wire delays and off-chip misses are going to grow in importance. Traditional architectures such as private and shared last-level caches provide a reasonable performance but they are not able to extract the maximum performance/watt of the memory hierarchy. Several proposals have appeared to find a compromise between these two configurations. Most of them use a banked shared L2 cache and distribute its capacity according to the requirements of each node [7,10,15,17,19]. In this type of configurations data does not necessarily have to be close to the requester, and because of that they may have a higher latency and network traffic.

On the other hand, some proposals have appeared, that in addition to the previous requirements, also try to reduce the access latency by placing the data close to the requesting node [3,5,8,9]. Some of these configurations use private caches and, on an eviction, forward the block to another cache. This technique, known as Spilling or N-Chance Forwarding [6], tries to dynamically adjust the program share of aggregate cache capacity depending on its activity.

However, existing configurations that use this technique forward all the evicted blocks to random nodes and this solution may not be optimal for most cases. Depending on the network, access latencies to the opposite side of the die may be extremely large which in turn can reduce performance and increase network usage. Also, depending on the application that is being executed spilling may not be necessary and could be switched off to reduce traffic and power consumption. In this paper we present two novel spilling techniques that spill the blocks to neighboring nodes and select whether to spill or not depending on the expected reuse of blocks. This paper makes the following contributions:

– We present two power-efficient spilling techniques suitable for both the centralized and distributed cooperative caching, the Distance-Aware Spilling and the Selective Spilling.
– We evaluate the proposed spilling techniques under two types of network for chip multiprocessors: a mesh and a bidirectional ring.

2 Background

We have tested our power-efficient spilling techniques with the Distributed Cooperative Caching (DCC) [9] scheme. In this configuration, every processor has its own private L1 and L2 caches to allow a low latency access to the data. To reduce off-chip accesses the Distributed Cooperative Caching uses an on-chip directory distributed across the chip in Distributed Coherence Engines (DCE). The DCEs are responsible of maintaining coherence for a portion of the address space and allow to know if a block is already in a cache of a different node without accessing memory. Figure 1 shows the DCC memory structure.

Addresses in the Coherence Engines are mapped in an interleaved way in order to distribute DCC requests across the network and avoid bottlenecks. This distribution implies that tag entries are allocated just in one DCE depending

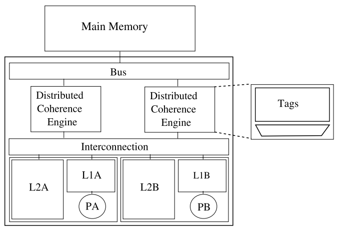

Fig. 1. DCC Memory Structure

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 will be necessary. However, a global directory is not enough to efficiently use the cache space. Although it allows to use all the cache data, it does not allow to allocate cache blocks in other nodes. Because of that, the Distributed Cooperative Caching also implements the N-Chance Forwarding mechanism [6] for replacements. When a block is evicted from an L2 the corresponding DCE 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 that tracks the number of times that is forwarded. When the counter reaches a threshold N, the block is evicted from the chip; however 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. In our configuration N is set to 1 since a replication control is already employed and further spilling may degrade performance by evicting newer blocks. The destination node for the spilled blocks is selected randomly.

3 Power-Efficient Spilling Techniques

As it has been shown, the N-Chance Forwarding mechanism is able to take advantage of the unused cache space with private caches. However, random spilling of all the evicted blocks can introduce unnecessary network traffic by forwarding to far nodes or by forwarding blocks that are not going to be reused. This extra traffic is going to increase the overall power consumption of the memory hierarchy and degrade its performance. We present two techniques to reduce power consumption without degrading performance, the Distance-Aware Spilling and the Selective Spilling.

3.1 Distance-Aware Spilling

Although a random selection of the destination node for spilling techniques is a good method to distribute the blocks across the chip, the reuse information of spilled blocks shows interesting optimization opportunities.

Figure 2 shows the percentage of spilled blocks being reused by the evicting node for the SPEC OMP benchmarks. As we can see most of the benchmarks

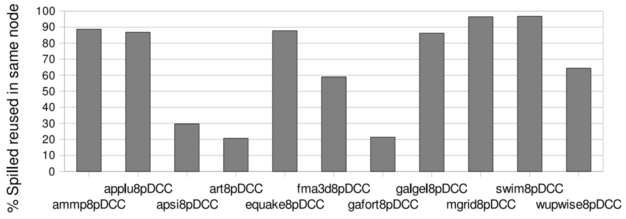

Fig. 2. Spilled blocks being reused by the evicting node

reuse evicted blocks in the same nodes that previously spilled them. If random spilling is used, data from one CPU in a corner of the chip may end up in the opposite corner and then is probably going to be reused by the original node. In a 4x4 mesh, this means traversing 6 hops per message. In recent architectures like the Intel Larrabee chip multiprocessor [16] this effect is even increased since a ring topology is used. In this case the maximum number of hops would be 8 for a 16 core configuration. This data transfers are going to increase the network traffic unnecessarily, increasing also the energy consumption and access latency.

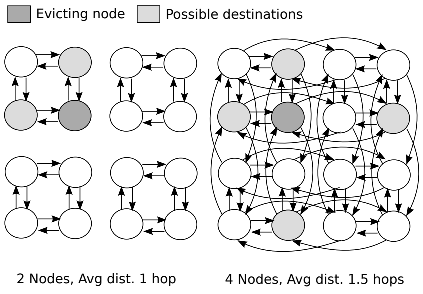

Fig. 3. Distance-Aware Spilling node assignment in a mesh network

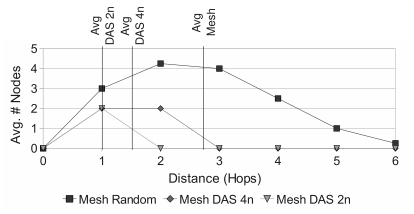

Fig. 4. Average distance to destination nodes in a mesh network

We propose a new spilling technique that aims to reduce the distance between the nodes involved in the spilling. In this case, we assign a set of fixed destinations for each node. Figure 3 shows the simulated mesh structure and the spilling destinations of each node represented by the arrows. For example, the sample evicting node (in dark grey) has 4 arrows departing from it that indicate the possible destination nodes (in light grey) in the 4 nodes configuration. These destinations are selected using a round-robin policy. As we can see, node assignments distribute the spilled blocks uniformly across the chip so every cache receives blocks from the same number of nodes.

Figure 4 shows the average number of nodes at each distance for the random and the two proposed Distance-Aware spilling policies. The figure also shows the average distance for these configurations. We can see that the average destination 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. Proposed distributions also have the advantage of having the same distance to destination nodes for all evicting nodes, while in the random distribution it depends on the position of the evicting node (middle, side or corner).

Figure 5 shows the assignment for a ring network. In this case, the benefit of the proposed technique is much higher since the average node distance is 4.27 for the random distribution (as shown in Figure 6). Our proposed policies, however, have the same average destination distance that we showed for the mesh; 1 hop for the 2 destinations configuration (DAS2n) and 1.5 hops for the 4 destinations configuration (DAS4n).

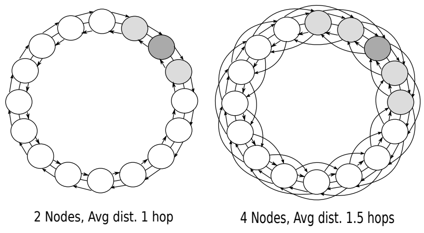

Fig. 5. Distance-Aware Spilling node assignment in a ring network

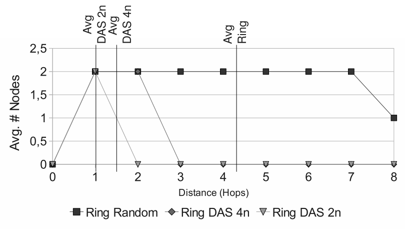

Fig. 6. Average distance to destination nodes in a ring network

Hardware requirements of the Distance-Aware spilling are very low, since only a round-robin arbiter per node is required for the 2 or 4 available nodes. The main limitation of this technique, is that there is a trade-off between the number of destinations (available cache space) and the distance to these caches. In case of having highly unbalanced memory requirements between threads, those requiring more cache space are not going to be able to use all of it, and this may reduce the performance. However, if the memory requirements are more balanced, our technique will be able to reduce the access latency and the network usage.

3.2 Selective Spilling

Another interesting optimization opportunity for the spilling mechanism comes from the fact that not all applications are going to benefit of the extra cache space provided by the N-chance forwarding technique. Therefore it would be interesting to have an adaptive mechanism that allows spilling only when blocks are expected to be reused.

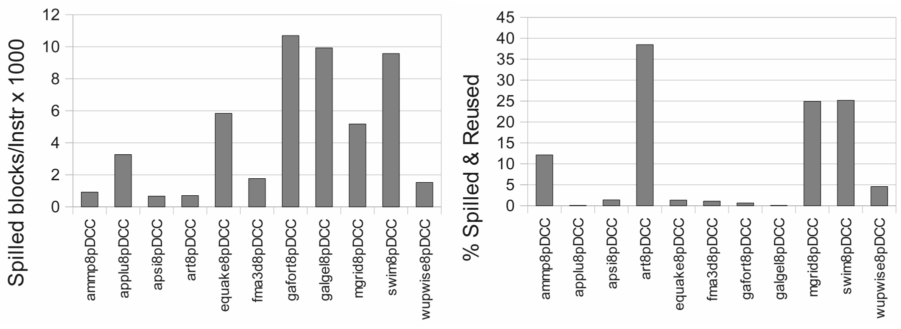

Fig. 7. Spilling characterization of benchmarks

Figure 7 shows the percentage of blocks that are reused after being spilled. It is possible to see that while is interesting to keep the spilling ability for applications like *Art*, other applications like *Gafort* do not make reuse of spilled blocks and have a high number of evictions/forwardings. These type of applications are going to insert a high amount of unnecessary network traffic and to reallocate blocks that are not going to be reused.

Our Selective Spilling mechanism decides whether to spill or not depending on the reuse of the previously spilled blocks. Our technique spills all the evicted blocks during a period of time C_t where a counter in each node keeps track of all the blocks that are being reused by that node. After this period of time every node decides if it is useful to spill or not and keeps this policy during $9*C_t$ cycles. The common spill period is followed by all nodes to be able to detect program phase changes. In our simulations we have used a C_t of 100k cycles and spilling is allowed if 5 blocks have been reused in this period of time. The period of time C_t and the threshold number of blocks have been determined empirically to provide a good performance with a reasonable overhead. The extra hardware required to implement this technique is only one counter of the reused blocks per node. It is possible to know if a block was spilled or not by the extra bit indicator that all configurations with the N-chance forwarding mechanism already have.

4 Experimental Setup

We have evaluated our proposed framework with Simics [11], a full-system execution-driven simulator extended with the GEMS [12] toolset that provides a detailed memory hierarchy model. We have added a power model of the network and the memory hierarchy based on Orion [18] to evaluate the energy efficiency of our proposal. Power from the cores has been estimated from similar configurations in the literature [13] since all configurations use the same processors and thus energy consumption is going to be similar. Energy efficiency has been measured in $MIPS^3/W$, which is an ED^2P formulation. The energy x delay^2 metric is the most appropriate metric for high performance computers according to Brooks et al. [2] and allows a voltage-and-frequency invariant power-performance characterization. Table 1 shows the values for the most important configuration parameters.

Our simulations have been done with ten pairs of benchmarks of the SPEC OMP2001 workload set with reference input sets, running simultaneously in half of the processors each. This means that in our 16 core configuration each benchmark is going to have 8 threads. Threads are allocated together so one half of the die runs the first benchmark and the other half the second. Benchmarks have been characterized by the number of evictions, reuse, and if reuse is made by the evicting node. Then benchmarks of table 1 have been selected to have all possible types of behaviors.

In all 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. Results are normalized respect to the Distributed Cooperative Caching with random spilling configuration. The evaluated cache organizations are; a Shared Cache,

Table 1. Configuration Parameters and Simulated Benchmarks

Parameter	**Value**	**Parameter**	**Value**
Number Processors	16	Block size	64 bytes
Instr Window/ROB	16/48 entries	L1 I/D Cache	16 KB, 4-way
Branch Predictor	YAGS	L2 Cache	256 KB, 8-way
Technology	70 nm	DCE size	36 KB, 8-way
Frequency	4 GHz	Hop Latency	3 cycles
Voltage	1.1 V	Link BW	16 bytes/cycle
Network Type	Mesh with 2 VNC	Memory Bus Latency	250 cycles

Benchmark	Spill/Instr	Reuse	Spill Reuse
Swim	High	High	High
Gafort	High	Low	Low
Ammp	Low	High	High
Art	Low	High	Low
Apsi	Low	Low	Low

Private Caches, Cooperative Caching (CC) [3] with a Coherence Engine capable of doing 2 transactions per cycle, Distributed Cooperative Caching (DCC) [9] with 1 DCE for each node/processor with 8-way associativity, two configurations of the Distance-Aware Spilling mechanism spilling to 2 neighboring nodes (DCC_DAS2n) and spilling to 4 neighboring nodes (DCC_DAS4n), the Selective Spilling (DCC_SS5) with C_t of 100k cycles and a spilling threshold of 5 reused blocks and a configuration with the Distance-Aware spilling and the Selective spilling together (DAS4n_SS5) spilling to 4 neighboring nodes, a C_t of also 100k cycles and a spilling threshold of 5 reused blocks.

5 Results

5.1 Mesh Network

The first network where we have studied our Power-Efficient Spilling techniques is the mesh.

Figure 8 shows the performance, energy efficiency and network activity of all the studied configurations. It can be seen that the proposed 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 data being reused. The combined solution shows a reduction of 26.6% of the network traffic. All these improvements are performed while keeping the same performance of the DCC with random spilling (the best performing configuration).

The second plot of Figure 8 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. Proposed techniques, further improve this technique an extra 4% (DAS2n). This

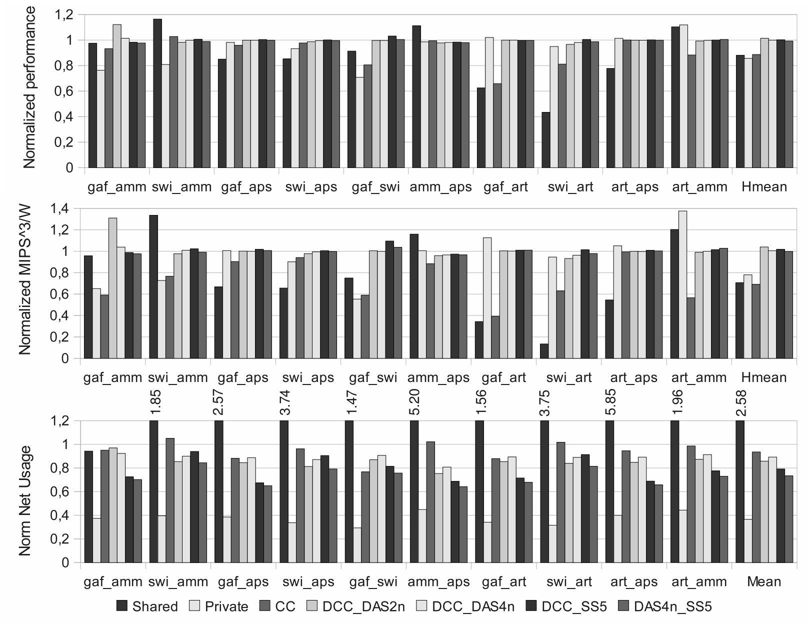

Fig. 8. Normalized performance, energy efficiency and Network Usage over DCC_ Random Mesh

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 the proposed techniques.

5.2 Ring Network

In this section we present the results obtained for a network with the same configuration parameters stated in Table 1 but changing the network topology to a bidirectional ring (as in the Larrabee chip multiprocessor [16]).

In this type of network, as it was shown before, the penalty of accessing far nodes is increased. 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 9 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

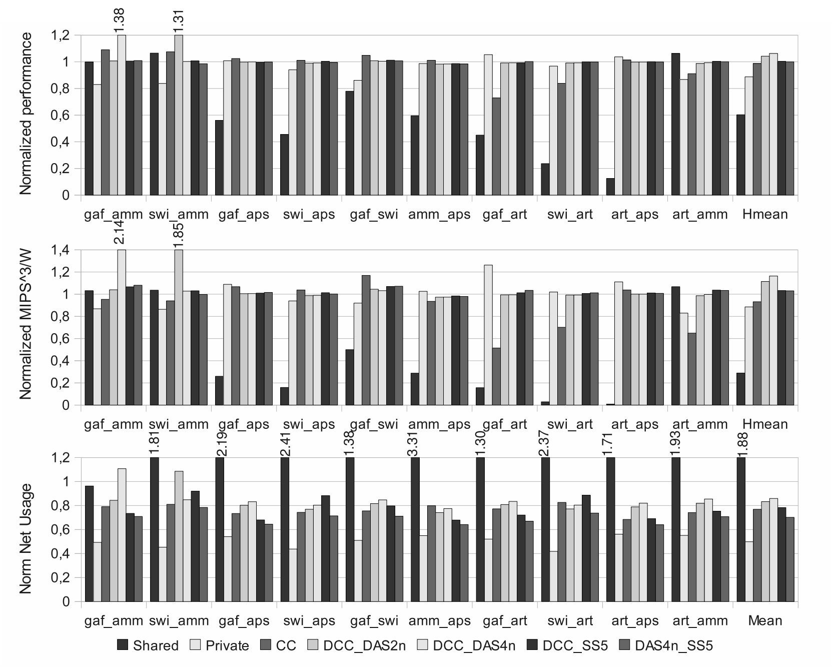

Fig. 9. Normalized performance, energy efficiency and network usage over DCC_ Random Ring

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 proposed configurations can be seen in the second graph of Figure 9. 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 caches 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 and that would probably increase the overall power had we considered the memory controller energy consumption.

5.3 Spilling Reuse

Finally, Figure 10 shows the percentage of spilled blocks that are reused later. It can be seen that random spilling makes very small reuse since all evicted

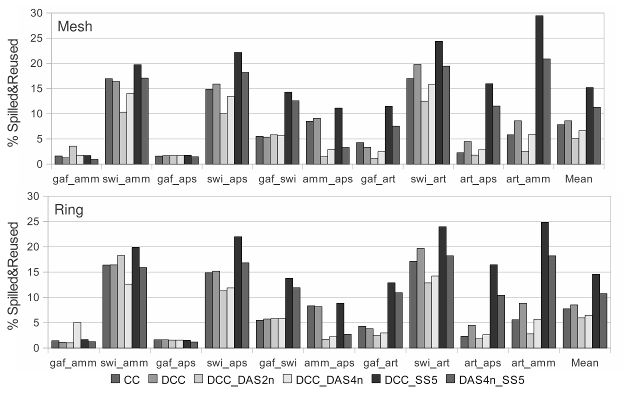

Fig. 10. Percentage of spilled blocks being reused

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_ammp benchmarks. The combined configuration (DAS+SS) also improves the random configuration in both cases while reducing the distance to the destination nodes. The trade-off between number of nodes to spill and distance can also be seen. 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.

6 Related Work

Many studies have appeared trying to find an optimal memory hierarchy for chip multiprocessors. Some of them use spilling as a sharing mechanism and can benefit of the proposed techniques [3,8,9]. However, several other configurations have appeared that also try to optimize the cache usage. This is the case of Utility-Based Cache Partitioning [15] and Adaptive Shared/Private Caches [7], that dynamically partition the last-level cache but do not try to reduce latency by allocating blocks in the closer nodes and are not scalable due to their centralized nature. The 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. Several proposals have also 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 Adaptive Set Pinning [17], a technique to reduce inter-processor misses by assigning a replacement ownership to every set. Qureshi [14] 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.

There has been also work done to make a distance-aware memory hierarchy, like Victim Replication [19], that proposes a new replacement mechanism to reduce miss latencies. This technique allows replication of blocks to local L2 caches if there is spare place to reduce latency but under heavy load conditions behaves as a normal shared cache configuration. CMP-NuRapid [5] from Chisti et al. replicates blocks in closer caches if the block is accessed several times. This proposal, however, is not scalable because blocks are found via snoop requests. Beckmann et al. proposed the Adaptive Selective Replication mechanism [1] that optimizes the level of replication dynamically to reduce access latency.

7 Conclusions

This paper has shown that the N-chance forwarding mechanism can be improved in terms of power and network usage while retaining its performance advantages for chip multiprocessors. Compared to Random Spilling, the proposed Distance-Aware Spilling technique provides an energy efficiency improvement in mesh and, specially, in ring networks (with even a small performance speed-up). In the latter, ED2P (MIPS^3 /W) is increased by 16% on average, network usage is reduced by 14% while performance increases 6%. At the same time, the Selective Spilling technique is able to avoid most of the unnecessary spilling in applications with low reuse reducing network traffic by an average of 22%. A combination of both techniques allows to reduce network usage by 30% on average without degrading performance, allowing a 9% increase of the energy efficiency. Hence, these techniques show to be very suitable for next generation chip multiprocessors with a high number of nodes, where the on-chip network is going to have a greater influence in the overall performance and energy efficiency.

Acknowledgements

This work has been supported by the Generalitat de Catalunya under grant 2009SGR1250 and the Spanish Ministry of Education and Science under grant TIN2007-61763.

References

1. Beckmann, B., Marty, M., Wood, D.: Asr: Adaptive selective replication for cmp caches. In: MICRO-39: 39th IEEE/ACM Int. Symp. on Microarchitecture (2006)
2. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
3. Chang, J., Sohi, G.S.: Cooperative caching for chip multiprocessors. In: ISCA 2006: 33rd Int. Symp. on Computer Architecture, pp. 264–276 (2006)

4. Chang, J., Sohi, G.S.: Cooperative cache partitioning for chip multiprocessors. In: ICS 2007: 21st Int. Conf. on Supercomputing, pp. 242–252 (2007)
5. Chishti, Z., Powell, M., Vijaykumar, T.: Distance associativity for high-performance energy-efficient non-uniform cache architectures. In: MICRO-36: 36th IEEE/ACM Int. Symp. on Microarchitecture, pp. 55–66 (2003)
6. Dahlin, M., Wang, R., Anderson, T., Patterson, D.: Cooperative caching: Using remote client memory to improve file system performance. In: OSDI 1994: 1st Conf. on Operating Systems Design and Implementation, pp. 267–280 (1994)
7. Dybdahl, H., Stenstrom, P.: An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors. In: HPCA 2007: 13th Int. Symp. on High Performance Computer Architecture, pp. 2–12 (2007)
8. Herrero, E., González, J., Canal, R.: Elastic cooperative caching: An autonomous dynamically adaptive memory hierarchy for chip multiprocessors. In: ISCA 2010: 37th Int. Symp. on Computer Architecture (2010)
9. Herrero, E., González, J., Canal, R.: Distributed cooperative caching. In: PACT 2008: 17th Int. Conf. on Parallel Architectures and Compilation Techniques, pp. 134–143 (2008)
10. Huh, J., Kim, C., Shafi, H., Zhang, L., Burger, D., Keckler, S.: A nuca substrate for flexible cmp cache sharing. In: ICS 2005: 19th Int. Conf. on Supercomputing, pp. 31–40 (2005)
11. Magnusson, P., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. Computer 35(2), 50–58 (2002)
12. Martin, M., Sorin, D.J., Beckmann, B., Marty, M., Xu, M., Alameldeen, A., Moore, K., Hill, M., Wood, D.: Multifacet's general execution-driven multiprocessor simulator (gems) toolset. SIGARCH Comput. Archit. News 33(4), 92–99 (2005)
13. Monchiero, M., Canal, R., Gonzalez, A.: Power/performance/thermal design-space exploration for multicore architectures. IEEE Trans. Parallel Distrib. Syst. 19(5), 666–681 (2008)
14. Qureshi, M.: Adaptive spill-receive for robust high-performance caching in cmps. In: HPCA 2009: 15th Int. Symp. on High Performance Computer Architecture (2009)
15. Qureshi, M., Patt, Y.: Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In: MICRO-39: 39th IEEE/ACM Int. Symp. on Microarchitecture, pp. 423–432 (2006)
16. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P.: Larrabee: a many-core x86 architecture for visual computing. In: SIGGRAPH 2008: ACM SIGGRAPH 2008 papers, pp. 1–15 (2008)
17. Srikantaiah, S., Kandemir, M., Irwin, M.J.: Adaptive set pinning: managing shared caches in chip multiprocessors. In: ASPLOS XIII: 13th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pp. 135–144 (2008)
18. Wang, H.S., Zhu, X., Peh, L.S., Malik, S.: Orion: a power-performance simulator for interconnection networks. In: MICRO-35: 35th IEEE/ACM Int. Symp. on Microarchitecture, pp. 294–305 (2002)
19. Zhang, M., Asanovic, K.: Victim replication: maximizing capacity while hiding wire delay in tiled chip multiprocessors. In: ISCA 2005: 32nd Int. Symp. on Computer Architecture, pp. 336–345 (2005)