Evaluation of Snoop-Energy Reduction Techniques for Chip-Multiprocessors

Magnus Ekman, Fredrik Dahlgren* and Per Stenström

Department of Computer Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{mekman, pers}@ce.chalmers.se

**Ericsson Mobile Platforms*
Nya Vattentornet
SE-221 83 Lund, Sweden
fredrik.dahlgren@emp.ericsson.se

Abstract

Chip multiprocessors (CMPs) have become an interesting micro-architectural style for high-end systems as well as low-power systems. While power-performance tradeoffs differ in these systems, a high power consumption can lead to devastating power densities in the former and a reduced operating time in the latter owing to limited battery capacity.

In this paper, we focus on the energy wasted in the snoopy cache protocols that keep the L1 caches in CMPs consistent. Previous studies have focussed on the energy wasted by snoop accesses in the private caches in SMP systems and found that it can be a big fraction of the total energy. We apply two techniques - serial snooping and Jetty - that were developed for SMP servers and see if they can lead to energy savings in a CMP. We find that the techniques are not well suited for a CMP and analyze why. Serial snooping does not work well because all caches have to be searched even if none can supply the data, which happens to be the case most of the time. Jetty, does not perform well because the snoop energy saved by the filtering is offset by the energy lost in the filters.

1. Introduction

Power dissipation and high-performance are two criteria of processing systems that are both of fundamental importance for an increasing number of computer systems [11]. In high-end systems, the main concern is the high power densities that may require special attention to cooling. On the other hand, in low-power systems, such as laptop computers, advanced phones and personal digital assistants (PDAs), the issue is the limited battery capacity. For the latter type of systems, techniques that provide rather high performance with small overheads in energy consumption are highly interesting.

Chip-multiprocessors (CMPs) [7] have gained a lot of interest as a micro-architectural style for future microprocessors for high-end systems [8, 14] as well as low-power systems [3, 5]. In reducing power consumption for low-power systems, CMPs offer several interesting opportunities: First, a multiprogrammed workload or a parallel program with enough thread-level parallelism can be run on several voltage- and frequency-scaled cores, leading to a lower power consumption than a uni-processor with the same performance. Secondly, as shown in [6], a simple in-order pipeline is more energy-efficient than an out-of-order superscalar pipeline which speaks in favor of CMPs with simple processor cores. Finally, in modes of operation where only a small fraction of the performance potential is requested, all but one of the simple and power-effective processors can be completely powered down.

Unfortunately, the amount of power that can be saved by parallelism is limited by the power wasted by inter-thread communication. In a CMP, communication is typically carried out through cache coherence actions across the private caches. In such protocols, *all caches* do a tag-lookup on a global (cache miss or upgrade) request, which waste energy. In fact, our measurements on a simulation model of a CMP driven by applications from SPLASH2 yield that more than 83% of all coherence actions are useless in the sense that they do not lead to any response. They can also lead to serious losses in the energy consumed in the L1 caches; our results indicate losses between 8% and 36% in a 16-way CMP.

We provide a comparative evaluation of two previously proposed techniques to reduce snoop-induced power in multiprocessors: serial snooping [12] and Jetty [10]. These two techniques were aimed at SMP-servers with two levels of private caches. Apart from analyzing them in a new context, i.e., chip-multiprocessors, this paper also provides detailed insights into what limits the gains of the techniques.

One observation is that serial snooping is ineffective because in most of the cases all caches have to be checked

before it can be concluded that they cannot respond to the request. The second observation is that since the private caches are typically smaller in a CMP, the reduction of the energy through the Jetty is to a large extent outweighed by the energy consumed in the Jetty.

The rest of this paper is organized as follows. Section 2 describes the baseline system and motivates why it should be possible to remove snoop-induced tag-lookups. Section 3 describes the techniques that are evaluated and how they are adapted to be used in a CMP. Section 4 describes the experimental methodology followed by Section 5 that describes the results. Finally Section 6 concludes the paper.

2. Baseline System and Motivation

We consider a CMP with private separate instruction and data L1-caches and a shared L2-cache like Hydra [7] according to Figure 1. The cache coherence protocol is a MOESI snoopy protocol [13]. To save energy, the L1-caches are searched first and the L2-cache is only accessed if the snoop action fails in all L1-caches. If a block is found in another L1-cache, a cache-to-cache transfer is used since it is cheaper to access an L1-cache than the L2-cache in terms of both performance and energy consumption. We assume that each L1-cache has dual tags so that the processor does not have to stall when a snoop request is issued from another processor.

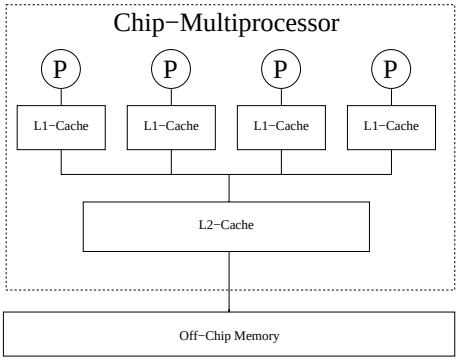

FIGURE 1. Baseline CMP architecture. All processors have local L1-caches and share an on-chip L2-cache.

Results in [12] indicate that as many as 32% of all snoop broadcasts, that are generated by loads, miss in all other caches meaning that they must be serviced by the next level of the memory hierarchy. This is despite the fact that they assume large private L2-caches. Results in [10], which assume smaller caches than [12], indicate that an even higher percentage (79.6%) of all snoop broadcasts miss in all other caches. In our system we have much smaller caches and our results show that as many as 83% miss in all other caches. This means that most of the snoop-induced tag-lookups just waste energy and encourage attempts to decrease the snooping activity.

3. Description of the Jetty and the Serial Snooping Schemes

In this section we describe the two techniques that are previously proposed, that try to limit snoop actions in a multiprocessor to save energy. Both of them are meant for an SMP environment and hence have not been evaluated for a CMP. Therefore we also describe how they have to be adapted to make sense in a CMP environment. Section 3.1 describes the Jetty. Section 3.2 describes the Jetty in a CMP. Section 3.3 describes Serial Snooping and Section 3.4 describes Serial Snooping in a CMP.

3.1 The Jetty Scheme

Jetty [10] is a small structure attached to each cache that filters out useless snoop accesses. Instead of doing a tag-lookup directly, the Jetty is first checked. In many cases, the information in the Jetty can tell if it is any use to do the tag-lookup or not. Since the Jetty is much smaller than the cache, the hope is that the Jetty-lookup results in significant energy savings. However, Jetty was proposed to save energy in a system where the processors have private L1- and L2-caches. There, the Jetty is used to filter out tag-lookups in the much bigger L2-caches and therefore the Jetty can afford to consume a reasonable amount of energy. An interesting issue is whether this holds in the CMP architectural framework we focus on.

There are three kinds of Jetties presented in [10]: exclude-Jetty, include-Jetty and hybrid-Jetty. The exclude-Jetty exploits locality in the snoop access stream, in the sense that if a block has been recently snooped, it is likely that the same block will be snooped again. An exclude-Jetty is a small cache structure connected to every cache, that records the blocks that were recently snooped and did not exist in the cache. On an incoming snoop, the exclude-jetty is first checked. If an entry for the block is present in the exclude-Jetty, it is guaranteed that the block does not exist in the cache and a tag-lookup in this cache can be avoided. A new entry needs to be inserted in the exclude-Jetty on every snoop-induced tag-lookup that misses. An entry needs to be searched for, and removed if found, each time a block is inserted into the cache. Formally, the exclude-Jetty keeps a subset of the blocks that are not locally cached.

The include-Jetty, on the other hand, keeps a superset of the blocks that are cached; if there is a miss in the include-Jetty, it is guaranteed that the block is not locally cached

and a tag-lookup can be avoided. Figure 2 shows an implementation of an include-Jetty. It consists of arrays of counters and arrays of p-bits. As seen in the figure, address bits are used to index the arrays to see what counter and what p-bit to access. When a block is loaded into the cache, the counters that are indexed by the address are increased and the p-bits are set. When a block is evicted from the cache, the counters are decreased and if the decreased value is zero, the p-bit is cleared. To see if a block may be in the cache, only the p-bits for the address have to be checked. If any of the p-bits that are indexed by the address is zero, the block can not be in the cache and no tag-lookup has to be done.

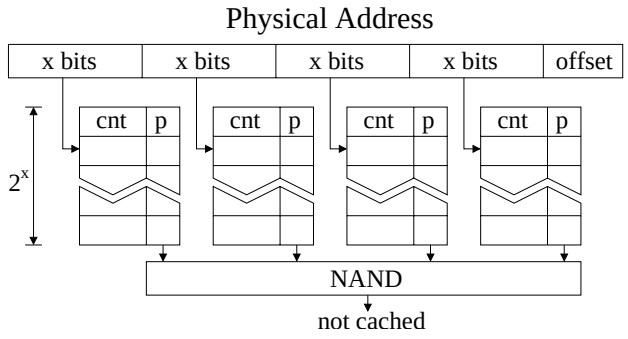

FIGURE 2. Implementation of include-Jetty.

The hybrid-Jetty is simply a combination of an exclude-Jetty and an include-Jetty that are checked in parallel. However, the net effect can be better than just adding the coverage of the exclude-Jetty and include-Jetty together since the include-Jetty filters some of the updates of the exclude-Jetty.

3.2 Adapting the Jetty to a CMP

Moshovos et al. [10] studied various include-Jetty configurations. One thing that they all have in common is that all address bits (except block offset) are used to index the cnt-arrays. Actually, sometimes the bits overlap. The problem with the system studied in this paper is that the caches are much smaller, and thus the Jetties must be smaller too. With a 32-bit physical address and a 5-bit block-offset we have to cover 27 bits. One solution would be to have four 128-entry arrays. Each array would require 7 bits in the address which gives 28 (4x7) address bits covered, yielding one overlapped bit. However, four 128-entry arrays require a considerable amount of energy compared to the caches. We are aware of that only the p-arrays (which are cheap to access) are checked on a snoop and thus do not consume a lot, but to keep the Jetties updated, the local cnt-arrays need to be accessed on each cache-miss, and this consumes energy.

An alternative would be to just cover some of the address-bits, but it is hard to predict if the Jetty would still work. It is reasonable to expect the behavior to deteriorate but it is hard to know if this will happen gradually or if it will break down abruptly. This is investigated in Section 5.2. The exclude-Jetty does not have the same problems. Certainly, it is not possible to have such big exclude-Jetties as in [10], but it is not a problem to resize them to make sense in a CMP.

The key interesting question we will address in the experiments is whether the reduction of the snoop-energy will be higher than the increase in energy by the Jetty mechanisms themselves.

3.3 The Serial Snooping Scheme

The serial snooping scheme is based on the assumption that if a miss occurs in one cache, it is possible to find the block in another cache without having to check all the other caches. If it is present in more than one of the other caches, the probability is high that the block will be found in a nearby cache. Even if the block is only present in one cache, it is still probable that the block will be found by just checking half the number of caches. Instead of broadcasting the snoop-transaction to all of the processors in parallel, the caches are checked serially in a system based on serial snooping. This technique only works for snoops that are induced by a read miss, since in these cases it is sufficient to retrieve the block from one cache. On a write, an invalidation transaction needs to be broadcast to all processors in the system and serial snooping cannot be used. (Actually it is possible to alter the scheme so that snooping is done serially on writes too. The termination criterion in that case is that the block shall be found as EXCLUSIVE or MODIFIED. However, since [12] does not deal with this we have not investigated it.)

The baseline system assumed in the paper that introduces serial snooping [12], is a hierarchical one with the nodes organized as a binary tree and presumably more than one processor per node. Further, it takes shorter time to just broadcast the transaction to a neighboring node than to the entire system. Thus, if a block is found in a neighboring node, performance is gained, which partly can compensate for the performance loss experienced when all nodes are checked serially without finding the block.

TABLE 1. System parameters.

Number of processors	4-16
Data-cache	8Kbyte 4-way - 32Kbyte 4-way
Block-size	32 bytes
Cycles to transfer a block from one cache to another	8
Cycles to transfer a block from L2-cache	20
Cycles to transfer a block from memory to L2-cache	100
Cycles per searched cache for serial snooping	1

3.4 Adapting Serial Snooping to a CMP

As mentioned above, the serial snooping scheme was applied to a hierarchical system with different propagation delays depending on which node that was accessed. We don't see why one would build a CMP with between eight and 16 processors hierarchically, and thus the conditions with respect to timing are different. In our implementation of the Serial Snooping scheme, the nearest neighbors are first accessed one cache at a time in a wrap-around fashion by treating the first and the last processor as neighbors.

For example, if processor 2 in an 8-processor system issues the request, the caches are checked in the following order: 3, 1, 4, 0, 5, 7, 6. In our timing analysis we assume that each cache that is searched adds a fixed delay to the miss-handling time. In the paper it is suggested that the L1- and L2-caches are checked serially but since this is already done in our baseline system that possibility is already exploited.

4. Experimental Methodology

We use Simics [9] to simulate a CMP memory system according to Figure 1 with between 4 and 16 single-issue processor cores. We have also incorporated mechanisms to model Jetty and Serial Snooping.

4.1 Architectural Timing Model and Applications

We model single-issue processor cores and assume that one instruction takes one clock cycle if there is a cache hit. Contention on the bus is modelled and the transactions are assumed to be atomic. We do not run an operating system so the applications do not migrate. We assume separate data- and instruction-caches and that no snooping is done in the instruction-caches. Therefore we only model the data-caches. Our assumptions as far as the architectural model are listed in Table 1.

We drive our experiments with the following set of applications/kernels. One source is the following set of four SPLASH2-benchmarks [16]: FFT, Raytrace, Water, and Barnes. We use the default input datasets that are recommended in the paper. We also use a parallelized version (mpeg2sliced_improved) [1] of mpeg2decode from the MPEG Software Simulation Group, which decodes the standard movie flwr_015.m2v.

4.2 Energy Model

During the simulation, statistics counters record the number of operations that take place in the memory system that affect the energy consumed in the L1-caches and in the energy saving structures. These are then multiplied with the energy cost associated with each operation. The cache configurations are calculated with Cacti [15]; a tool that attempts to make an optimal cache configuration with respect to latency. The power-model for a 0.18 CMOS process from Wattch [2] is used, which models the decoder, the word-line driver, the bit-line discharger and the sense-amplifier. We do not model static energy consumption since we are primarily interested in ultra-low power devices. According to [4], the leakage current is more than three magnitudes lower for an ultra-low power process than a ultra-high speed. This means that the approximation that the static power consumption is negligible is still valid.

We assume that the caches are accessed in a two-stage manner so that tags are checked in parallel but that only the accessed data is read. In Wattch, it is assumed that both data and tags are read in parallel since a more aggressive system with higher performance is assumed, in contrast to the low-power system that we study.

5. Simulation Results

This section describes the simulation results. Section 5.1 describes how much snoop-induced energy that can be removed in the system, while Section 5.2 deals with sizing the Jetty. Section 5.3 shows how much snoop energy that is removed by the techniques and Section 5.4 describes why an exclude-Jetty does not work in a CMP. Section 5.5

investigates the total energy-saving achieved, and finally Section 5.6 looks at the performance aspects of the techniques.

5.1 Snoop-induced L1 Cache Energy

Figure 3 shows the fraction of the energy consumed in the L1-caches that is caused by snooping, for 4-, 8-, and 16-way CMP systems. While this fraction is low for the 4-way CMP, it becomes a significant part for 8-way CMPs ranging from 5% to 22%. As expected, the amount of energy consumed increases with the number of processors and varies between 8% and 36% for a 16-way CMP. In the rest of the paper, we will use an 8-way CMP per default. Table 2 shows the L1 hit rates for all of the applications for that system and it is clearly seen that a low hit rate increases the energy consumed by snooping; for example FFT and Radix both have low hit rate and high snoop energy consumption.

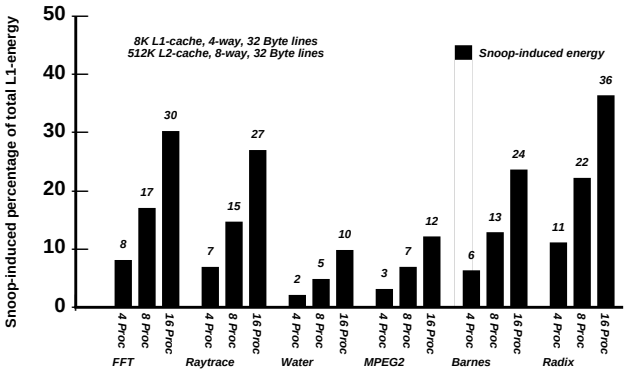

FIGURE 3. Snoop induced percentage of the entire L1-cache energy consumption.

TABLE 2. Hit rates for an 8-processor system 8K caches

Benchmark	L1 Hit-rate
FFT	93.7%
Ray trace	95.1%
Water	98.6%
MPEG	97.9%
Barnes	95.7%
Radix	90.9%

Let us now investigate to what extent the snoop-induced energy can be wiped out. Remember that a snoop access, resulting in a tag lookup in one cache, is useless if it does not lead to any response. An example of this is on a load if a block exists as SHARED in some caches and OWNED in another, which leads to the cache in the OWNED state supplying the data. The caches in SHARED state do not supply data and do not change coherence state, which means that the tag-lookups are useless. Table 3 shows the fraction of the snoop broadcasts that hit in 0, 1, 2, 3, 4, 5, 6 and 7 caches in an 8-way CMP.

A vast majority of the snoop broadcasts – 83% on average – miss completely. For another 9% of the snoop broadcasts, only one cache holds the block. Clearly, we would expect that there is a great potential to save snoop-induced energy. All the tag lookups that miss are potentials to save energy, and in addition to these, tag lookups that do not result in a state-transition or data supply can be removed.

TABLE 3. Percentage of snoops that hit in the caches.

Benchmark	0 hit	1 hit	2 hits	3 hits	4 hits	5 hits	6 hits	7 hits
FFT	99.4%	0.2%	0.1%	0.1%	0.1%	0.1%	0%	0%
Raytrace	79.1%	11.1%	3.7%	2.4%	1.8%	1.2%	0.6%	0.2%
Water	57.1%	28.5%	9.4%	2.3%	1.0%	0.8%	0.6%	0.2%
MPEG	96.4%	1.7%	0.3%	0.2%	0.3%	0.4%	0.5%	0.1%
Barnes	53.3%	17.5%	12.0%	8.6%	5.2%	2.5%	0.8%	0.1%
Radix	99.0%	0.7%	0.2%	0.1%	0%	0%	0%	0%
Average	83.3%	8.7%	3.7%	2.0%	1.2%	0.7%	0.4%	0.1%

5.2 Sizing the Include-Jetty

As mentioned in Section 3.2 it is hard to predict the behavior of the include-Jetty when making it smaller to be reasonable in a CMP. To get a feeling for this we have simulated different Jetty-organizations to see what happened when the number of bits that were used to index the arrays (and thus the size of the arrays), were decreased. The results are shown in Figure 4.

.

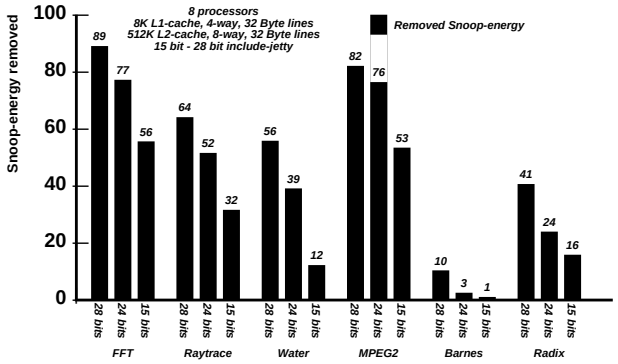

FIGURE 4. Tests of different include-Jetty configurations.

The figure shows how much snooping energy that was removed by different Jetty-organizations. Note, however, that the energy consumed by the Jetties themselves is not included. As one can see, the removed snoop-energy decreases gradually, and not abruptly, when going from 28 (covering all address bits) to 24 and finally to 15 bits. This means that it should be possible to use an include-Jetty in a CMP. It is just a matter of sizing it reasonably. After studying different Jetty-organizations we have chosen to use a hybrid-Jetty with the parameters shown in Table 4.

TABLE 4. Jetty parameters

Include-jetty	3 tables with 32 entries
Exclude-jetty	32 entries, 4-way associative

To decide these parameters we have also included the energy cost of the Jetty itself, and tested a few configurations. Figure 5 shows the total energy consumed by the L1-caches and the energy consumed by the include-Jetties themselves for various sizes of the Jetties.

Clearly, it is not reasonable to cover all the 28 bits in the address since the Jetties consume to much energy. A similar study was done for the exclude-Jetty which is more described in Section 5.4.

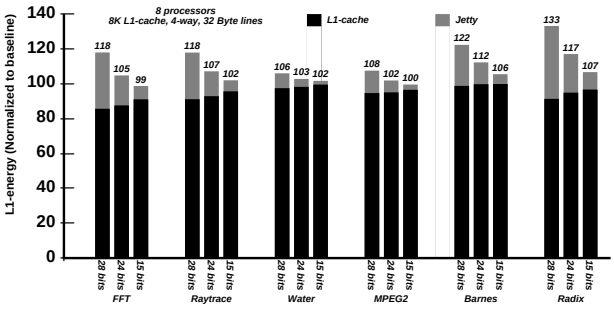

FIGURE 5. Total energy results for various include-Jetties.

5.3 Removed Snoop Energy

Figure 6 shows how much of the snoop-induced energy that is removed by the techniques, and the ideal bars show how much that is possible to remove.

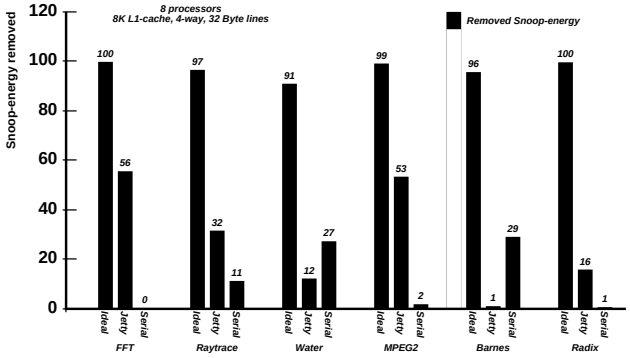

FIGURE 6. Comparison of how much of the snoop-induced energy that is removed with the different techniques.

The applications seem to express very few of the properties that are needed for the Serial snooping scheme, especially MPEG, Radix and FFT. This is not very surprising, since these three benchmarks all have very little shared data, which is demonstrated in Table 3. As a result, a snoop request has to be propagated through the entire system before it is determined that the block is not found and hence no energy is saved.

The Jetty scheme seems to have more potential. In two of the applications it can cut more than 50% of the snoop-induced energy. Measurements that we have done show that almost all of the reduced energy is due to the include-Jetty. The exclude-Jetty hardly filters any tag-lookup. The next section describes why the exclude-Jetty does not work in a CMP.

5.4 Trashing of Exclude-Jetty in a CMP

As Table 3 shows, hardly any snoop-induced tag-lookups hit in the remote caches. Since the exclude-Jetty is updated on a miss, almost every snoop will lead to an update of the exclude-Jetty. The snoops are induced because of cache-misses or because a processor wants to change the state of an already loaded block to MODIFIED. The misses are caused by cold/capacity/conflict-misses and coherence-misses. If the major part of the snoops is induced due to cold/capacity/conflict-misses, and each of these transactions will update the exclude-Jetty, then the Jetty will experience the same update-pattern as an L2-cache. Thus, we can not expect a better hit-rate in the exclude-Jetty than in an L2-cache with room for as many blocks as there are entries in the exclude-Jetty. An exclude-Jetty with 32 entries would experience the same hit-rate as an L2-cache of 1KB (32x32), which is a smaller cache than the first level cache. The problem for the exclude-Jetty in a CMP is that due to the small caches, most of the snoop-induced tag-lookups are due to conflict/capacity-misses. These updates trash the exclude-Jetty, and destroy the possibilities to catch producer-consumer behavior for example. Because of these problems we do not consider an exclude-Jetty in the rest of the paper.

Even if we had not had any problems with trashing there is another reason that the exclude-Jetty would not work well, namely that we do not use sub-blocking This is necessary to get a good hit rate when there is little or no sharing according to [10], where they assume that the cache blocks are 64 bytes and divided into two 32 byte sub-blocks. The cache coherence protocol works on the sub-block level while the exclude-Jetty records blocks on the block-level. This means that when two cache misses occur on consecutive blocks, the second miss will probably be filtered by the exclude-Jetty since the first miss recently updated the exclude-Jetty with an entry covering the two sub-blocks. However, this assumes that there has not been many updates to the Jetty between the two misses.

5.5 Total Energy Results

The reduced snoop-energy of the techniques is interesting, but the total energy that is consumed is the bottomline number, since that takes the overhead associated with the techniques into account. For the include-Jetty, the counters need to be updated every time a block is read into or thrown out of the cache. The present-arrays also need to be accessed. On each snoop broadcast, all remote present-arrays are checked to try to eliminate a tag-lookup. The Serial snooping technique is approximated as having no energy-overhead, while it in practice has a bus overhead. Figures 7 and 8 show the total energy results for the different systems.

The results show that Jetty saves energy in some cases and in some it does not. An alternative approach would be to make the Jetties even smaller, but that affects the coverage of the Jetty too much. Serial snooping mostly saves a few per cent energy. However, this is when we approximate the technique to have no overhead. The ideal bars indicate that there is still plenty to cut.

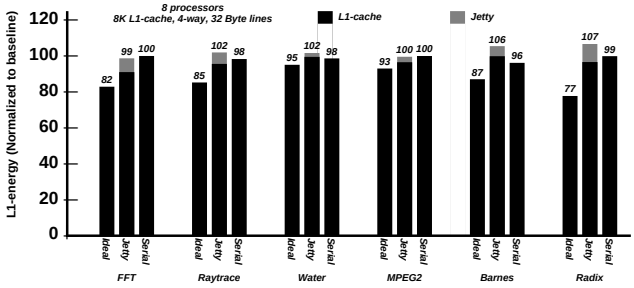

FIGURE 7. Energy comparison for an 8-processor system.

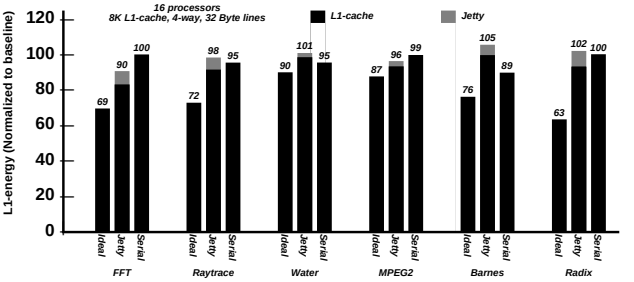

FIGURE 8. Energy comparison for a 16-processor system.

5.6 Performance Results

Jetty is approximated as having no performance penalty, which is true if the time it takes to check the Jetty does not increase the bus cycle time. Serial snooping increases the time to serve a cache miss, and the time is dependent on where in the system a block is found. Table 5 shows the performance penalty for a system with Serial snooping as well as some other properties.

Since we have a rather rough timing model of the processor core, these numbers should not be interpreted as exact numbers but more to get a feeling for the properties of the benchmarks that affect the performance of the serial snooping scheme. Overall, there is a significant performance overhead. As more processors are added, performance will deteriorate and serial snooping will probably

need to be done hierarchically for systems with 16 processors or more. However, this would decrease the energy saved.

The performance penalty is dependent on the number of processors, the number of accesses to the bus, the amount of these accesses that are writes, and the amount of shared data in the system. This is best understood by studying the applications. Radix has a very small amount of shared data, which means that the snoops that are reads mostly propagate to all the caches. It also has a very low hit-rate (90.9%) in the L1-caches, which means that there is much snooping in the system. However, according to Table 5, the amount of write misses compared to the amount of read misses is rather high. Since Serial Snooping is not performed on writing, the miss-handling time will not increase for the write misses and therefore the performance penalty is not that high that the miss-rate and amount of shared data would imply.

FFT on the other hand, has a higher hit-rate (93.7%) and little shared data, but a lower amount of write misses compared to read misses than Radix, which gives FFT slightly higher performance loss. Barnes (hit rate 95.7%), which experiences severe performance loss has a very low amount of write misses compared to the amount of read misses. Even though Barnes has rather much shared data, and hence the penalty for each read access is not as big as if there was no shared data, the increased miss handling time gives a big performance penalty.

6. Conclusion

The snoop induced tag-lookups consume a significant amount of energy in the L1-caches in a CMP. Since most of these lookups are useless, it is an interesting target for power reduction.

In this paper we have presented a comparative evaluation of two previously proposed techniques to cut the snoop-induced energy. Both these techniques were proposed in another context, i.e., SMP systems, where significantly bigger private caches are used. By evaluating them in a CMP context, where the private caches are much smaller, an immediate consequence is that more snoop-induced tag-lookups miss and hence more of the snoop-induced tag-lookups are useless.

The most important observations of our study are the following: We first find that up to 36% of the energy consumed in the L1 caches in a 16-way CMP is attributable to snooping and more than 83% of this energy could be conserved if the outcome of the snoop action was known.

Serial snooping does not manage to cut much energy because most of the time, no caches will be able to respond which means that all caches will be searched. For Jetty, a significant portion of the snoop-energy is cut but this saving is outweighed by the energy lost in the Jetty. Despite our effort to study the impact of a range of Jetty parameters, we did not manage to find any design point where a significant saving of energy was observed.

To conclude, reducing snoop energy in CMPs appear to be an important problem to solve. Yet, previous techniques to address the same problem in an SMP environment are quite useless. Thus, there is room for considerable innovation and we are currently investigating a solution to this problem.

Acknowledgement

This research has been supported by a grant from the Swedish Foundation for Strategic Research under the ARTES/PAMP program.

TABLE 5. Performance issues for an 8-processor system 8K caches

Benchmark	Performance loss Serial	% of misses that are reads	% of misses that are writes
FFT	5.8%	78.2%	21.8%
Raytrace	11.7%	91.1%	8.9%
Water	3.2%	94.1%	5.9%
MPEG	8.5%	90.1%	9.9%
Barnes	21.2%	97.6%	2.4%
Radix	4.3%	34.8%	65.2%

References

[1] A. Bilas, J. Fritts and J. P. Singh. Real-Time Parallel MPEG-2 Decoding in Software. *Proceedings of the 11th International Parallel Processing Symposium*, pages 197-203, April 1997.

[2] D. Brooks, V. Tiwari and Margaret Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. *Proceedings of the 27th International Symposium on Computer Architecture*, pages 83-94, June 2000.

[3] D. Courtright. MIPS32 M4K Core for Multi-CPU Applications. *Embedded Processors Forum*, April 30, 2002.

[4] K. Diefendorff. TSMC Sets Sights on #1. *Microprocessor Report*, June 5, 2000.

[5] M. Edahiro, S. Matsushita, M. Yamashina, N. Nishi. A Single-Chip Multiprocessor for Smart Terminals. *IEEE MICRO Magazine*, pages 12-20, July-August 2000.

[6] R. Gonzales and M. Horowitz. Energy Dissipation In General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits*, pages 1277-1284, September 1996.

[7] L. Hammond, B Hubbert, M Siu, M Prabhu, M Chen and K Olukotun. The Stanford Hydra CMP. *IEEE MICRO Magazine*, pages 71-84, March-April 2000.

[8] IBM. Power4 System Design for High Reliability. *Hot Chips 13*, August 2001.

[9] P. S. Magnusson, F. Larsson, A. Moestedt, B. Werner, F. Dahlgren, M. Karlsson, F. Lundholm, J. Nilsson, P. Stenström, and H. Grahn. SimICS/sun4m: A virtual workstation. *Proceedings of the USENIX 1998 Annual Technical Conference. USENIX Association*, pages 119-130, June 1998.

[10] A. Moshovos, G. Memik, B. Falsafi and A. Choudhary. JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers. *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pages 85-96, January 2001.

[11] T. Mudge. Power: A first class design constraint. *Computer*, vol. 34, no. 4, April 2001, pages 52-57

[12] C. Saldanha and M. Lipasti. Power Efficient Cache Coherence. *Workshop on Memory Performance Issues, in conjunction with ISCA*, June 2001

[13] P. Sweazey and A. J. Smith. A Class of Compatible Cache Consistency Protocols and Their Support by the IEEE Futurebus. *Proceedings of the 13th International Symposium on Computer Architecture*, pages 414-423, May 1986.

[14] M. Tremblay. MAJC: Microprocessor Architecture for Java Computing. *Hot Chips*. August 1999.

[15] S. Wilton and N. Jouppi. An Enhanced Access and Cycle Time Model for On-chip Caches. *WRL Research Report 93/5, DEC Western Research Laboratory*, 1994.

[16] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh and A. Gupta. The SPLASH-2 programs: Characterization and methodological considerations. *Proceedings of the 22th International Symposium on Computer Architecture*, pages 24-36, June 1995.