TLB and Snoop Energy-Reduction using Virtual Caches in Low-Power 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

In our quest to bring down the power consumption in low-power chip-multiprocessors, we have found that TLB and snoop accesses account for about 40% of the energy wasted by all L1 data-cache accesses. We have investigated the prospects of using virtual caches to bring down the number of TLB accesses. A key observation is that while the energy wasted in the TLBs are cut, the energy associated with snoop accesses becomes higher. We then contribute with two techniques to reduce the number of snoop accesses and their energy cost. Virtual caches together with the proposed techniques are shown to reduce the energy wasted in the L1 caches and the TLBs by about 30%.

Categories and Subject Descriptors

C.5.3 Microcomputers---Microprocessors

General Terms

Performance, Design

Keywords

low-power, CMP, snoop, virtual caches

1 INTRODUCTION

Performance demands of low-power systems, such as lap-top computers, PDAs, and advanced mobile phones are increasing but must be accommodated within the constraints of the limited battery capacity. As a base for meeting this need, we consider chip multiprocessors (CMPs) based on several simple cores, which easily can be shut down independently due to the modularised design. However, to take advantage of this potential power-saving capability and yet achieve the full performance potential, assumes low performance and energy losses in the memory system.

A CMP memory system often uses private L1 caches attached to each processor which are connected via a bus to a shared L2 cache using a snoopy cache protocol to maintain consistency [7]. Assuming physically-addressed L1 caches, each memory access triggers a TLB access and an L1 cache lookup. In addition, misses in the L1 cache or modifications of shared blocks result in a snoop action leading to a tag-lookup in *all* L1 caches even if a cache will not respond to the request. We measured the energy wasted in the L1 caches as a result of local and snoop actions and the energy wasted in the TLBs for multiprogrammed and multithreaded workloads on an 8-way CMP simulation model. We found that TLB energy losses are typically 20-25% of the energy wasted in the L1 caches and snoop actions account for 5-10% of the L1 energy losses.

ISLPED'02, August 12-14, 2002, Monterey, California, USA.

In this paper, we investigate the prospects of reducing TLB energy losses in a CMP by considering virtually addressed L1 data-caches (see, e.g.,[3]) and focus for the first time on their energy costs. A first observation is that while the energy cost associated with the TLBs can be considerably reduced, it comes at the cost of higher energies for the snoop accesses. We also contribute with two techniques that reduce the number of snoop accesses and the cost of each of them. The first technique is a novel mechanism, called *Page Sharing Table*, which is an auxiliary structure attached to each TLB. By keeping track of the sharing set of each page, we show that it can almost completely eliminate all useless snoop accesses when there is little or no sharing. This results in a considerable L1 cache energy reduction. In order to support the use of multiple page sizes, we also propose a technique, called the *Page Size Information Scheme,* that uses information of the size of the page which the block belongs to in order to limit the number of tag lookups needed on each snoop access. Our techniques reduce the energy wasted in the L1 caches and in the TLBs by about 30%.

2 BASELINE CMP SYSTEM

We consider a CMP with private separate instruction and data L1-caches and a shared L2-cache similar to Hydra [7] (see Figure 1). The cache coherence protocol is a MOESI snoopy protocol, but the evaluation is applicable to other snoopy protocols, such as MSI and MESI. 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 more effective to access an L1-cache than the L2-cache. We assume that each L1-cache has a dual tag store so that the processor does not have to stall when a snoop request is issued from another processor. We do not consider the instruction-caches. However, the proposed system essentially eliminates snooping overhead related to instruction fetches.

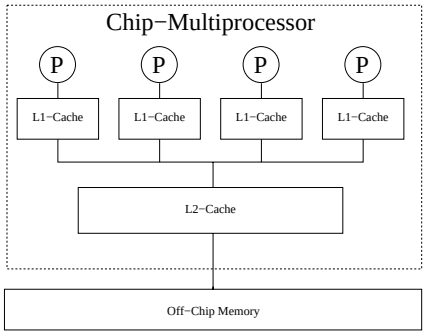

FIGURE 1. Baseline CMP architecture.

While the baseline uses physically-addressed (or physical) caches, we now consider the implications on power consumption using virtually-addressed / virtually-tagged (or virtual) caches. The potential gain of using virtual caches is that no address translation is needed on cache hits which reduces the energy consumed by the TLB since the TLB is now only accessed on cache misses. However, virtual caches introduce other problems. Because of page limitations, we will here focus on the most essential issues related to the focus of this study.

A first issue is the synonym problem, where multiple virtual addresses map to the same physical address. An effective approach in a multiprocessor with one level of private caches in a snoop-based system is to have two sets of tags, one virtual and one physical [6]. Only the virtual tag is accessed on L1 cache accesses at cache hits. At a miss in the virtual tag store, the physical tag store is checked for a synonym mapping. In the case of a hit, i.e. the block is present at a different virtual address, the block is moved to a location corresponding to its most recent virtual address. Often, and dependent on page and cache sizes, only an update of the virtual tag is required rather than an additional block copy. In order to reduce the performance impact, the synonym check based on the physical address is carried out in parallel with the bus-request. At the rare occasions when a synonym block is present in the cache, the bus transaction is cancelled.

The physical tags are also checked at bus snoops in order to determine a block's presence and status in the cache. Section 3 further discusses organizational tradeoffs of the physical tag-store and shows why a snoop access can waste more energy than in physical caches.

A second issue is the alias problem where different processes have identical virtual addresses mapped to different physical addresses. We solve this by flushing only the virtual tag store at context switches, as our experience indicates such flushes to be few and less sensitive for application-processing systems (as opposed to real-time sensitive control systems). Accesses to blocks still present in the cache will be found by the physical tag-store interrogation at cache misses presented above, and will lead to slightly fewer long-latency / high-energy misses than if the complete cache had been flushed. An alternative approach is to extend the virtual tag-store with a process identifier (PID), but our experiments indicate this to increase the L1 cache energy by between 3% and 8%, depending on system parameters. Using a PID and not flushing the virtual tags also implies invalidating a virtual tag when another process loads the corresponding physical block.

Overall, while virtual caches can reduce the energy wasted in the TLB, an important source of energy losses are the snoop accesses. Therefore, the next section introduces two techniques that aim at reducing this energy cost.

3 SNOOP-ENERGY REDUC. TECHNIQUES

As mentioned above, the snoop-induced energy is increased in a system with virtual caches. Earlier work [11,10] has shown that as many as 30-80% of the snoop accesses are indeed useless and could be removed.

The PST scheme is based on the intuition that there exist a fair number of pages that are not shared. A page is said to be loaded in a processor if at least one block that belongs to the page is loaded in the private cache. For non-shared pages, blocks are not subject to coherence actions and snooping overhead could be eliminated. While the scheme is particularly good for single-threaded applications (with no sharing), we will see that they work well also for parallel programs.

A unit called Page Sharing Table (PST) is attached to each processor. This unit keeps track of which pages are currently used by the processor. For each such page, the unit keeps a sharing vector that indicates if the other processors also share the page. This sharing vector is broadcast on a separate bus, called *sharing vector bus,* with as many lines as the number of processors, on a snoop-broadcast action. By reading this sharing vector, the other caches know whether they need to do a tag-lookup to check for the block or not. Since the PST keeps track of whether there is any block cached from a particular page, no information about the page means that the local physical tag-store need not be accessed when there is a miss in the virtual tag-store. A system with PSTs is shown in Figure 2.

To keep the sharing vectors updated, a PST in one node must ask PSTs in other nodes if they share the page, when a new entry is loaded. If they share the page, they must also update their sharing vectors, since a new processor now shares the page. To be able to ask if the other processors share the page, it must be possible to look for an entry in the PST with the help of the physical address. The other PSTs indicate that they share the page through switching their corresponding line on the sharing vector bus.

Since the way to decide whether a page is shared is based on looking for the page entry in the PSTs of the other processors, it is necessary that inclusion is maintained between the PST and the L1 cache. That is, if an entry is not loaded, then it is guaranteed that no blocks belonging to that page are loaded. If inclusion is not maintained, a page can be marked as not shared even though blocks that belong to the page are loaded in another cache. Inclusion is maintained by loading a page entry into the PST the first time a block on the page is accessed. Conversely, when a PST entry is evicted, one must make sure that all blocks belonging to that page are evicted from the local cache.

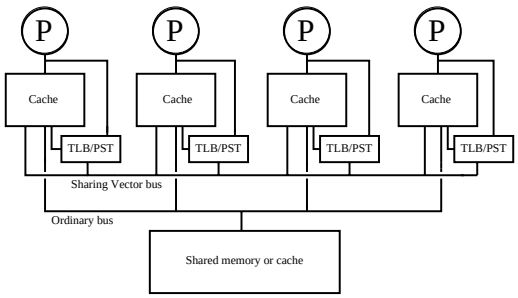

FIGURE 2. System with PSTs.

To avoid energy-costly invalidations in the local cache, we adopt the following scheme: Each entry in the PST has a counter that is incremented when a block that belongs to the page is inserted into the cache and decremented when the block is evicted. When a page entry needs to be evicted from the PST, an entry with a counter that is zero is chosen as a victim. If no entry with a zero counter exists, the PST scheme is shut down and snoop accesses are handled as in the baseline system. However, when other nodes update their PSTs and ask if the PST has a certain entry, it always indicates that it has, since inclusion is not maintained when it is shut-down and there is no way to guarantee that no block from the page is loaded in the cache. When the PST is turned on again, the cache must however be flushed. We will explain what turns it on later.

When an entry is replaced in the PST, the sharing vector is checked to see if the other processors share the page. A broadcast on the bus notifies the processor nodes that share the page that the page is not used by the current processor anymore. The processors that are affected by the broadcast are notified using the sharing vector bus.

Under certain conditions the PST will not work well, namely if there is practically no locality at all in the memory references. If almost no blocks that are accessed are located on the same pages, the PST-miss rate will approach the cache miss rate. If this happens, the traffic to keep the PSTs updated will increase and waste more energy than the PST saves. This behavior also implies that there will often not be any PST entry with a counter value of zero and thus the PST is switched off.

A mechanism that can switch on the PST is also needed, since it is possible that a process has some sections with little locality which switches off the PST, followed by a section with much locality where the PST should be active. The way to detect when the PST should be turned on is based on comparing the cache hit-rate with the TLB-hit rate. This ratio describes how many accesses we have on a page before an entry needs to be replaced. Since the PST saves energy on cache misses but wastes energy on its own misses due to update costs, this ratio in some sense describes if there will be a net gain or loss.

Since the TLB keeps entries for all pages that are accessed by the processor, it makes sense to integrate the TLB and the PST. A sharing vector is associated with each TLB-entry. We also need to be able to access entries using the physical address, to be able to

check whether other nodes share a page. However, it must be possible to change the size of the PST independently from the TLB. Therefore, the PST also has a victim part which contains entries that have been evicted from the TLB. Figure 3 shows one possible implementation. The page size does not have to be the same in the PST as in the virtual memory system. If the pages in the PST are bigger, more than one entry in the TLB may correspond to one entry in the PST.

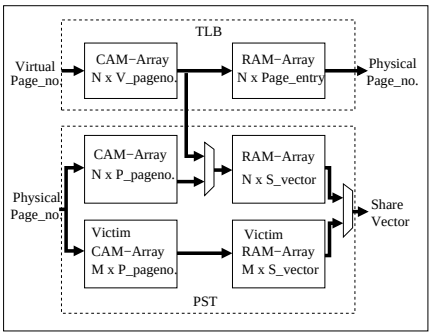

FIGURE 3. An implementation of the PST.

The mechanism that identifies when a PST should be switched on is based on an 8-bit counter for each PST. For each cache-miss the counter is incremented one tick. For each TLB-miss the counter is decremented by a preset number of ticks. The counter does not wrap around if it is already zero or 255. As long as the number of TLB-misses are close to the number of cache-misses the counter will stay close to zero, but when the TLB-miss rate decreases, the counter value increases. When it passes a threshold, the PST is switched on again. We have empirically found that oscillation is not a problem if the number of ticks that the counter is decremented by on a TLB-miss is chosen conservatively. (In our experiments, it is preset to 2.)

The problem of physical tag-store lookups of virtual caches is that virtual addresses might map differently to the cache than its corresponding physical address. This problem occurs when the cache size divided by its associativity is larger than the page size. For example, for a 16 kB direct-mapped virtual cache, and if the page size is 4 kB, a block's physical address can be mapped to four different cache lines depending on address mapping. As most processor architectures support different page sizes, the system must be designed to handle also the smallest page size. This implies that a 16 kB 4-way cache will need a 16-way physical tag-lookup for systems with a 1 kB minimum page size, such as in ARM.

Prior art proposes two classes of solutions that do not need to increase the associativity; either cache addressing conflicts are accepted leading to paired eviction which gives rise to additional cache misses, or by using page coloring or other virtual-to-physical mapping restrictions. The latter gives rise to lower cache- and/or MMU-performance, more complex handling of virtual pages, and becomes inherently hard for systems with a non-homogenous memory system.

Our proposed technique is based on the finding that performance-demanding applications rarely are dominated by access to the smallest page size. For larger pages, the required associativity of physical tag-lookup can be lower, given that the page size of the current access is known. We propose the *Page Size Information Scheme* (PSI) which simply transmits the page size together with the address on the bus. The page size is encoded and observe that it is only necessary to encode the sizes that are smaller than *cachesize/associativity*. The physical tag-store is modified so that it takes this page size into account when deciding possible places to check in the cache. For large pages this reduces the snooping cost to, in essence, that of physical caches.

4 EXPERIMENTAL METHODOLOGY

We use Simics [9] to model the baseline CMP system and the virtual cache extension according to Section 2. In addition, we also model systems based on the latter but extended with the PST and PSI schemes. We do not run an operating system so the applications are run with each thread tied to one processor. Our assumptions for the architectural parameters are as follows. We model CMPs with eight processors attached to 4-way 16 kB L1 caches with a 32-byte block size and the 8-way L2 cache is 512 kB big. The Data TLB and the PST contain 32 and 64 entries, resp. Page sizes range from 1 to 4 kB. We drive our experiments the following set of SPLASH2-benchmarks [12]: FFT, Raytrace, Water, Barnes and Radix. We use the default input data sets 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. In addition, we run a multiprogrammed workload based on applications from MediaBench [8].

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, the TLBs and in the PSTs. These are then multiplied with the energy cost associated with each operation. The power-model for a 0.18 micron CMOS process from Wattch [2] is used. We do not model static power 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 an ultra-high speed. This means that the approximation that the static power consumption is negligible is still valid.

We do not take into account the energy wasted by the bus. The activity on the wide data/address bus is expected to be decreased with our PST scheme because of fewer global write / invalidate requests, but the sharing vector bus adds to the energy losses. With the PSI Scheme a few bits are needed for the page size information (one bit in the system studied). To get an idea of how much the bus would impact the results we calculated how much energy that was wasted to drive a bus-line (including interference with neighboring lines) from one side of the chip to the other, with the driver sized so it would be possible to do this in a half clock-cycle. The energy for this was about 25% of the energy consumed by one (4-way associative) tag-lookup. Many times no lines need to be driven since there is no sharing, while the number of tag-lookups that are reduced are the same as the number of processors. As said in the previous section, the associativity of the physical tags often needs to be increased, resulting in even more saving while it does not affect the bus.

5 EXPERIMENTAL RESULTS

The energy distribution in the caches and TLBs in a CMP naturally depends on the size and associativity for both the caches and the TLBs. The left most bar for each benchmark in Figure 4 shows the distribution for our baseline system. The gray field in the middle of the bar is the energy consumed by snoop-induced tag-lookups that miss in the caches. Between that field and the darkest region is a small field that represents the energy consumed by snoop-induced tag-lookups that hit in the cache. According to the figure this field is zero for all benchmarks indicating that almost all snoops miss in all caches. The fraction of the energy that is wasted by snooping depends on the hit rate in the cache, since all misses result in a snoop transaction. For example, Radix with a hit rate of 93.0% suffer more from snooping than the multiprogrammed workload which has a hit rate of 99.3%.

The second bar shows the energy distribution for a system with virtual caches. The energy wasted by the TLB is drastically decreased. For this bar it is assumed that the associativity in the physical tag-store is four times the virtual associativity to be able to handle page sizes of 1K, to avoid conflicts that would lead to paired eviction. Hence, the energy wasted by snooping should be four times the snooping in the baseline system. On each miss the local physical tag is checked to find synonyms.The energy wasted

by this is included in the snooping section of the bar, which explains why the snooping energy is more than quadrupled for Radix. We can see that as long as the miss-rate is not very high, the increased snooping cost is lower than the energy saved in the TLB.

The third bar shows a virtual cache system with 1 kB pages using a PST, where each entry corresponds to a 4 kB page. The PST removes some of the snoop-induced energy. For the multiprogrammed workload the PST removes practically all snooping, since the different processes have no shared data and are located at different pages. For FFT, which has very little shared data, almost all of the snoop-energy is removed but the PST itself wastes some energy. This is due to low locality that leads to a lot of traffic to keep the PSTs updated. Radix has very little shared data but during a major section of the program the PST is turned off due to too little locality. This is why the scheme does not remove more. The same holds for Raytrace. MPEG2 removes approximately half of the snooping-energy even though the PST is turned on all the time. This has to do with false sharing on the page level.

With the Page Size Information Scheme (PSI) the associativity of the physical tag-store can often be as small as that of the virtual tag-store. The only exception is when *actual page size* < *cache size/associativity.* Our experience indicates the usage of 1 kB pages to be rare for performance-centric applications, but this is dictated by the system software. In order to illustrate the *potential* effects of PSI, we have run simulations where the actual page size is 4 KB for all pages in our benchmark applications. The fourth bar shows the impact on energy. In a real system, there will exist some pages of the minimum size, where the PSI scheme will not reduce the energy losses of some snoops, but the average snoop energy of the PSI scheme is expected to be close to the PSI bar of Figure 4.

The fifth bar (PSI/PST) shows a system with PST and PSI combined, for the scenario where all executed pages are at least 4 KB. Although this combined scheme provides the best overall result, the difference between PSI and PSI/PST is often small. However, for real scenarios where page sizes often are smaller than *cache size / associativity* (4 KB in our example), the PSI scheme alone would have less impact while the combined scheme will still perform at least as well as the PST alone. Overall, for systems where pages smaller than *cache size / associativity* might be common, the combined scheme is by far the most robust design point. Overall, the PSI/PST example eliminates most of the TLB and snoop-induced energy consumption, and leads to an energy reduction of the L1/TLB/snooping system by around 30% for all applications in the study.

6 CONCLUDING REMARKS

Other studies have also acknowledged the high energy losses associated with snoop accesses in shared-memory multiprocessors[10,11]. These techniques have been further studied in a CMP environment in [5].

We have investigated the option of using virtual caches in chip multiprocessors with a focus on its impact of energy losses in the TLB and in the L1 cache access path. We have found that while the energy losses in the TLB can be considerably reduced, synonym issues can make energy losses associated with snoop accesses higher. In order to reduce the energy losses associated with snoop accesses, we introduce the *Page Sharing Table* scheme that is able to remove a majority of the snoop accesses that would miss in the L1 caches anyway. In order to reduce the energy losses of the remaining snoop accesses in systems that support multiple page sizes, we introduced the *Page Information Scheme*. From our detailed simulations using multiprogrammed as well as parallel workloads, we find that the combination of these techniques result in a reduction of the energy losses in the L1 caches and the TLB by about 30% on average.

7 REFERENCES

[1] A. Bilas, J. Fritts and J. P. Singh. Real-Time Parallel MPEG-2 Decoding in Software. *Proc. of the 11th Int. 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. *Proc. of the 27th Int. Symp. on Computer Architecture*, pages 83-94, 2000.

[3] M. Cekleov and Michel Dubois. Virtual-Address Caches, Part 2: Multiprocessor Issues. *IEEE Micro*, pages 69-74, Nov/Dec 1997.

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

[5] M. Ekman, F. Dahlgren and P. Stenström. Evaluation of Snoop Energy-Reduction techniques for Chip-Multiprocessors. *Workshop on Duplicating, Deconstructing and Debunking, in conjunction with ISCA*, May 2002.

[6] J. Goodman. Coherency for Multiprocessor Virtual Address Caches, *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS-II)*, pages 72.81, 1987.

[7] L. Hammond et al., The Stanford Hydra CMP. *IEEE MICRO Magazine*, pages 71-84, March-April 2000.

[8] C. Lee, M. Potkonjak and W. H. Mangione-Smith. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. *Proc. of the Int. Symp. on Microarchitecture*, pages 330-335, 1997.

[9] P. S. Magnusson et al., SimICS/sun4m: A virtual workstation. *Proc. of the USENIX 1998 Annual Technical Conference. USENIX Association*, pages 119-130, June 1998.

[10] A. Moshovos et al., JETTY: Filtering Snoops for Reduced Energy Consumption in SMP Servers. *Proc. of the 7th Int. Symp. on High-Performance Computer Architecture*, pages 85-96, January 2001.

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

[12] S. C. Woo et al., The SPLASH-2 programs: Characterization and methodological considerations. *Proc. of the 22th Int. Symp. on Computer Architecture*, pp. 24-36, 1995.

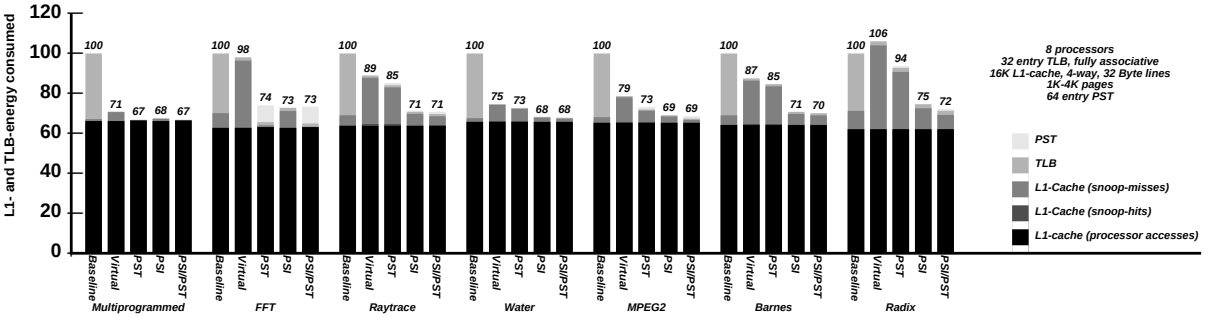

FIGURE 4. Energy distribution for the different systems for all benchmarks. (For explanation, see text.)