Memory System Energy: Influence of Hardware-Software Optimizations*

G. Esakkimuthu, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802

ABSTRACT

Memory system usually consumes a significant amount of energy in many battery-operated devices. In this paper, we provide a quantitative comparison and evaluation of the interaction of two hardware cache optimization mechanisms (block buffering and sub-banking) and three widely used compiler optimization techniques (linear loop transformation, loop tiling, and loop unrolling). Our results show that the pure hardware optimizations (eight block buffers and four sub-banks in a 4K, 2-way cache) provided up to 4% energy saving, with an average saving of 2% across all benchmarks. In contrast, the pure software optimization approach that uses all three compiler optimizations, provided at least 23% energy saving, with an average of 62%. However, a closer observation reveals that hardware optimization becomes more critical for on-chip cache energy reduction when executing optimized codes.

1. INTRODUCTION

Hardware and software techniques to reduce energy consumption have become an essential part of current system designs. Such techniques have particularly targeted the memory system due to the prevalent use of data-dominated signal and video applications in mobile devices. Various low power circuit techniques, energy efficient memory and cache architectures [2, 6] and power-aware compilation techniques [1] have been proposed. However, there is still much work to be done in understanding the interaction of hardware and performance based software optimizations and in evaluating their relative energy gains. This paper explores the interaction of hardware and software optimizations by considering the energy savings obtained when using energy-efficient cache architectures and compiler optimization techniques.

The rest of this paper is organized as follows. The hardware and software optimizations employed in this work are discussed in Section 2. Our experimental strategy and results are given in Section 3. Finally, Section 4 provides concluding remarks.

*This work was supported in part by grants from NSF, PDG and Sun Microsystems

ISLPED '00, Rapallo, Italy.

2. OPTIMIZATIONS

A variety of hardware optimizations have been proposed to reduce energy consumption. In this paper, we focus on two cache optimizations, namely, block buffering [6, 2] and cache sub-banking [6, 2], as the cache is one of the major energy consuming components in current processors. We choose these cache optimizations since the effectiveness of block buffering is influenced by software optimizations while that of sub-banking is not.

In the block buffering scheme, the previously accessed cache line (block) is buffered for subsequent accesses. If the data within the same cache line is accessed on the next data request, only the buffer needs to be accessed. This avoids the unnecessary and more energy consuming access to the entire cache data array. Thus, increasing temporal locality of the cache line through software techniques can save more energy. In the cache sub-banking optimization, the data array of the cache is divided into several sub-banks and only the sub-bank where the desired data is located is accessed. This optimization reduces the per access energy consumption and is not influenced by locality optimization techniques. We also evaluate cache configurations that combine both these optimizations. In such a configuration with block buffering and sub-banking, each sub-bank has an individual buffer. Here, the scope for exploiting locality is limited as compared to applying only block buffering as the number of words stored in a buffer reduces. However, it provides the additional benefits of sub-banking for each cache access.

The compiler optimizations evaluated (i.e., linear loop transformations, tiling,and unrolling) are known to improve the performance characteristics of programs enormously [9], but their impact on energy consumption needs further evaluation. Among the various high-level compiler transformations, we choose to target loop optimizations for two reasons. First, the multimedia and signal processing applications operate on multi-dimensional array structures that benefit from such optimizations. Second, these optimizations are widely used by commercial and academic optimizing compilers.

`Linear Loop Transformations:` The linear loop transformations attempt to improve cache performance, instruction scheduling, and iteration-level parallelism by modifying the traversal order of the iteration space of the loop nest. The simplest form of loop transformation, called loop interchange [9], can improve data locality (cache utilization) by changing the order of the loops. We therefore expect this optimization to particularly benefit the block buffering optimization discussed earlier.

`Loop Tiling:` Another important technique used to improve cache performance is blocking, or tiling [9]. When it is used for cache locality, arrays that are too big to fit in the cache are broken up into smaller pieces (to fit in the cache). Here again we expect a decrease in power consumed in memory, due to better data reuse after applying tiling. This optimization exploits temporal locality across multiple loops.

`Loop Unrolling:` In this technique outer loops are unrolled a certain number of times. The advantage of doing so is that it reduces the number of memory accesses. From the energy perspective, fewer accesses to the memory means less energy consumption. It should be mentioned that these optimizations are generally applied in conjunction with scalar replacement, which increases the number of data accesses. However, loop unrolling compensates this effect by minimizing the number of load/store operations in the inner loop positions. Although not addressed in this paper, it should be noted that these optimizations typically result in more complex array subscript functions and loop bounds which in turn can increase the overall energy consumption in the data path and instruction caches [7].

3. INFLUENCE OF OPTIMIZATIONS

In order to evaluate the effectiveness and interaction of the hardware and software optimizations, the C versions of benchmarks shown in Table 1 were used. All these codes are representative of the multi-dimensional array domain, the domain that many signal and video processing applications belong to. In this study, we zoom in on the `mxm` benchmark results when varying the different parameters and finally summarize the behavior across all benchmarks in the end. To determine the energy consumed by these codes, we obtained memory reference traces while executing the benchmarks using the cycle-accurate simulator [7]. These traces were then analyzed using a configurable memory system simulator that was built in-house. The memory system simulator allows the configuration of cache sizes, block sizes, associativities, write and replacement policies, the number of cache sub-banks and cache block buffers used. Also incorporated in our memory system simulator is the on-chip cache energy model proposed in [3] using 0.8 technology parameters [8], the off-chip main memory energy per access cost of the Cypress CY7C1326-133 chip and the I/O pad energy costs [5]. To evaluate the impact of compiler optimizations on the overall energy consumption, we used an extended version of a high-level compilation framework [4] based on loop (iteration space) and data (array layout) transformations.

3.1 Hardware Optimizations

The energy consumed by the `mxm` benchmark in the data cache with different configurations of block buffers and sub-banks (The number of block buffers being either 2, 4 or 8 and the number of sub-banks varying from 1 to 4) was studied for a 4K cache with various associativities. This result showed that increasing the number of sub-banks from one to two provides an energy saving of 45% for the data cache accesses. An additional 22% saving is obtained by increasing the number of sub-banks to 4. It must be observed that the savings are not linear as one may expect. This is because the energy cost of the tag arrays remains constant, while there being a small increase in energy due to additional sub-bank decoding. We found that for block buffering adding a single block buffer reduced the energy by up to 50%. This reduction is achieved by capturing the locality of the buffered cache line, thereby avoiding accesses to the entire data array. However, access patterns in many applications can be regular and repeating across a varied number of different cache blocks. In order to capture this effect, we varied the number of block buffers to two, four, and eight as well. We observed that, for the `mxm` benchmark, an additional 17% (as compared to a single buffer) energy saving can be achieved using four buffers.

`tomcat`	Specfp95	9	119	4,868,048
`nasa7/btri`	Specfp92	29	4,312	44,430,039
`nasa7/m m`	Specfp92	3	120	47,168,707
`nasa7/ penta`	Specfp92	9	114	2,274,945
`adi`	Livermore	6	241	6,062,860
`dtdt (aps)`	Perfect Club	17	1,605	42,119,337
`bmcm ( ss)`	Perfect Club	11	126	89,539,244
`psmoo (tfs)`	Perfect Club	1	204	16,955,980
`eflu (tfs)`	Perfect Club	5	297	12,856,306

Table 1: Programs used in the experiments.

We also found that using a combination of eight block buffers and four sub-banks, the energy consumed in 4K (16K) data cache could be reduced on an average by 88% (89%). Thus, such hardware techniques can reduce the energy consumed by processors with on-chip caches. However, if we consider the entire memory system including the off-chip memory energy consumption, the energy savings from these techniques amount to only 4% (15%) when using a 4K (16K) data cache. Thus, it may be necessary to investigate optimizations at the software level to supplement these optimizations.

3.2 Software Optimizations

We also studied the total energy consumption of the memory system after applying the compiler optimizations on a 4K data cache with block buffering or sub-banking. (In this paper, `t`, `l`, `u`, and `tlu` denote pure tiling, pure loop transformations, pure unrolling, and the concurrent application of all these three optimizations, respectively.) We find that the maximum energy gains of these optimizations are observed when using data caches with lower associativities (1-way and 2-way) and the entire memory energy reductions using a 4K (16K) direct mapped data cache range from 58-75% (63-79%) for the different optimizations. As the cache associativities increase, the influence of conflict misses reduces significantly, thereby, downplaying the importance of these locality optimizations. In fact, the energy could increase in such cases as a result of applying the tiling and loop transformation when they are accompanied by scalar replacement (i.e., due to increased number of memory references). Further, we find that the combination of all the

three optimizations when applied together achieves a maximum savings of 75% energy saving for the 4K cache configurations investigated.

The energy consumed in data caches after applying the compiler optimizations with block buffering or sub-banking was also studied.We find that the tiling and loop transformations, generally, increase the cache energy consumption due to the additional memory references introduced by scalar replacement. This effect is pretty dominant in the tiling optimization that results in a 186% increase in energy for 4K data caches on an average. However, this effect is overshadowed by the drastic reduction in the number of more energy consuming off-chip accesses to the main memory. We also observe that the unrolling optimization is suitable for reducing both the cache energy and the overall energy in the `mxm` code.

3.3 Combined Optimizations

It was found that when a combination of different software and hardware optimizations is applied, tiling performs the best among the three individual compiler optimizations applied in terms of memory system energy across different 4K cache configurations Since, we mentioned earlier that tiling increases the cache energy consumption. sub-banking and block buffering are of particular importance here. For the tiled code, moving from a base data cache configuration to one with eight block buffers and four sub-banks reduces the overall memory system energy by around 10%. Thus,

Further, we observed that the linear loop transformed codes exploited the block buffers better than the original code and other optimizations. For example, when using two (eight) block buffers in a 4K 2-way cache, the block buffer hit rate was 69% (82%) as compared to the 55% (72%) for the unoptimized `mxm` code. Thus, it is also important to choose the software optimizations such that they provide the maximum benefits from the available hardware optimizations.

So far, we have presented the results only for the `mxm` code to discuss the influence and interaction of the software and hardware energy optimization techniques. Figure 1 captures this influence for all the studied benchmarks. These results show that there is a significant potential for reducing energy through a proper combination of hardware and software optimization techniques. The pure hardware optimizations (eight block buffers and four sub-banks) provided up to 4% energy savings, with an average saving of 2% across all benchmarks. In contrast, the pure software optimization approach (`tlu`), provided at least a 23% energy saving, with an average of 62%. Further, a combination of hardware and software optimizations provides an average of 64% energy saving. It is thus observed that the compiler optimizations provide most of the savings in the memory system energy consumption, while hardware optimization can be critical if one focuses on on-chip cache energy consumption.

4. CONCLUSION

The goal of this study is to investigate the interaction and influence of hardware and software optimizations on the memory system energy. To achieve this goal, we selected two effective cache optimizations and three widely used compiler optimizations and experimented with nine multi-dimensional array codes. Our results show that even performance based compiler optimizations provide a significantly higher energy savings as opposed to those gained using the pure hardware optimizations considered. However, a closer observation reveals that hardware optimization become more critical for on-chip cache energy reduction when executing optimized codes.

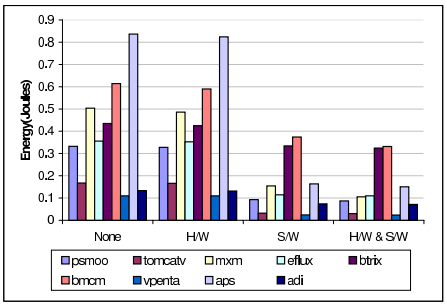

Figure 1: Energy consumed when using a 4K, 2 way data cache with a combination of hardware and software optimizations. None refers to the base hardware and original code; Software refers to base hardware and optimized code generated when all optimizations are enabled; Hardware refers to data cache with eight block buffers and four sub-banks, and original code; Hardware/Software refers to data cache with eight block buffers and four sub-banks and optimized code generated when all optimizations are enabled.

5. REFERENCES

[1] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandecappelle. Custom memory management methodology – exploration of memory organization for embedded multimedia system design. June, 1998.

[2] M. Kamble and K. Ghose. Reducing power in super-scalar processor caches using sub-banking, multiple line buffers and bit-line segmentation. In , pages 70–75, 1999.

[3] M. Kamble and K. Ghose Analytical energy dissipation models for low power caches. In , pages 143–148, 1997.

[4] M. Kandemir, A. Choudhary, J. Ramanujam, and P. Banerjee. Improving locality using loop and data transformations in an integrated framework. In Dallas, TX, December, 1998.

[5] W.-T. Shiue and C. Chakrabarti. Memory exploration for low power, embedded systems. Center for Low Power Electronics, Arizona State University, , September 1999.

[6] C. Su and A. M. Despain. Su and Despain, Cache design trade-offs for power and performance optimization: A case study. In , pages 63–68, 1995.

[7] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using SimplePower. In Vancouver, British Columbia, Canada, June, 2000.

[8] Wilton, S.E, and Jouppi, N., An enhanced access and cycle time model for on-chip caches, July 1994.

[9] M. Wolfe. . Addison-Wesley Publishing Company, 1996.