Dead-Block Elimination in Cache: A Mechanism to Reduce I-Cache Power Consumption in High Performance Microprocessors

Mohan G Kabadi, Natarajan Kannan, Palanidaran Chidambaram, Suriya Narayanan M Subramanian, and Ranjani Parthasarathi

School of Computer Science and Engineering
Anna University, Chennai - 600 025, India.
{mohan_kabdi,natarajan,palanidaran,mssnlayam,rp}@cs.annauniv.edu

Abstract. Both power and performance are important design parameters of the present day processors. This paper explores an integrated software and circuit level technique to reduce leakage power in L1 instruction caches of high performance microprocessors, by eliminating basic blocks from the cache, as soon as they are dead. The effect of this dead block elimination in cache on both the power consumption of the I-cache and the performance of the processor is studied. Identification of basic blocks is done by the compiler from the control flow graph of the program. This information is conveyed to the processor, by annotating the first instruction of selected basic blocks. During execution, the blocks that are not needed further are traced and invalidated and the lines occupied by them are turned off. This mechanism yields an average of about 11% reduction in the energy consumed by the I-cache for a set of the SPEC CPU 2000 benchmarks [16], without any performance degradation.

1 Introduction

In the present scenario, silicon area and power have become important constraints on the designers. New technical developments are being implemented to overcome the former constraint. The fabrication technology of VLSI circuit is steadily improving and the chip structures are being scaled down. But, the number of transistors on a chip is increasing at a higher ratio. Also, the drive towards increasing levels of performance has pushed the operating clock frequencies higher and higher, which has resulted in an increased level of power consumption [1].

Power has thus become important, not only for wireless and mobile electronics, but also for high performance microprocessors. It should therefore be considered a “first class” design constraint on par with performance [2].

There has been considerable work on low power processors as evidenced in the literature. Many of these have focused on reducing power/energy in the memory subsystem namely multi-level I-caches [3], [4], [5], [6], [7], [17], d-caches [6], [7], [8], [9], [17] and main memory [10], [11]. Memory subsystems, especially the on-chip caches have caught the attention of designers for the reason that, they are dominant sources of power consumption. Caches often consume 80% of the total transistor count and 50% of the area [12]. Hence cache subsystems have been the primary area of focus for power reduction.

Several techniques have been proposed to reduce the power dissipated by on-chip caches. These techniques can be grouped under (i) architectural alternatives adopting dynamic [5], [6] or static methods [13] and (ii) Software techniques using compiler support [3], [8]. In recent years, software techniques have been receiving more attention in building low power systems. The idea here is that the compiler can assist the microarchitecture in reducing the power consumption of programs either by giving explicit information on program behavior or by optimization. Ideas that have been explored include use of a reduced size cache/buffer to store inner loops [3], and the use of loop transformations to improve the performance of the program, thereby reducing the energy consumed [8]. However, these approaches are effective only for loop intensive programs. This paper presents a more generic, novel, compiler-assisted technique, wherein, selected lines of the cache are turned off under program control, resulting in reduction of power consumption. This approach takes care of both loop-intensive and non loop-intensive programs.

This paper is organized as follows: In section 2, a brief review of the related work relevant to the proposed approach is presented. In section 3, the proposed method, Dead-Block Elimination in Cache (DBEC) is described. In section 4, the hardware modifications required for the proposed method are outlined.

The experimental methodology and the results of the simulation are given in section 5 and section 6 concludes the paper.

2 Related Work

As mentioned in the previous section, compiler supported power minimization is a field of active research. A few of the techniques relevant to the proposed work are presented below. In the approach taken by Nikolaos Bellas et al. [3], a small L-cache similar to the Filter cache [13], is added between the CPU and the I-cache, for buffering instructions (basic blocks) that are nested within loops. However, the basic blocks within loops that contain function calls are not considered for placement in the L-cache. The profile data from previous runs is used to select the best instructions to be cached.

Another approach taken by Hongbo Yang et al. [8], is based on the fact that performance improvement causes a reduction in execution time and hence energy saving can be achieved. Here, the impact of loop optimizations on performance of program vs power are compared. Compiler optimizations namely loop unrolling, loop permutation, loop tiling and loop fusion are shown to improve program performance, which is correlated positively with the energy reduction.

Other techniques that have been explored are hardware-based or architecture-based. Gated-Vdd [4] is a circuit-level technique to gate the supply voltage and reduce leakage in unused SRAM cells. The DRI I-cache [5], dynamically reacts to application demand, and adapts to the required cache size during an application's execution. This work uses Gated-Vdd mechanism at the circuit-level, to turn-off the supply voltage to the cache's unused sections, thus reducing leakage power. The DRI I-cache integrates architectural and circuit level techniques to reduce leakage in an L1 I-cache.

On a similar line as that of DRI I-cache, in the approach used by Kaxiras et al [9], [17], parts of the L1 d-cache are switched off at a much finer granularity (i.e. line granularity) and without resizing. The key idea here is that, if a cache line is not accessed within a predefined fixed interval, the supply voltage to that line is turned off using the Gated-Vdd mechanism. This approach uses a static turn-off interval which is set on an individual application basis, to obtain optimal results.

In the approach followed by Huiyang Zhou et al. [6], a hardware counter called Line Idle Counter (LIC) is used, to keep track of the length of the period a line remains idle, before it gets turned off. The Mode-Control Logic (MCL) compares the LIC value to the turn-off interval stored in a Global Control Register. The miss rate and performance factor are dynamically monitored to set the Global Control Register, periodically, at the end of a statically set predefined interval.

3 Dead-Block Elimination in Cache (DBEC)

The work done in [4], [5], [6] has shown that, dynamically turning off sections of the I-cache and resizing it, results in significant leakage-power savings. Motivated by this approach, the current work explores the "turning-off" of the unused I-cache lines using a software-directed approach. In [5], [6] the time at which a line can be turned off is determined at runtime based on a saturating counter. The choice of the saturation value directly affects performance, as the saturation value is only an estimate of the non-usage of a line. Too large a value would result in reduced power saving, while too small a value would evict the line earlier from the cache, resulting in a miss and thus reducing performance. An optimal value has to be chosen based on the application. Even this application dependent static choice of the saturation value is only an estimate. However, more precise information on when a line is going to be "dead" can be directly obtained from the compiler. The compiler support for pointing out when a block is dead is used in the proposed approach.

The proposed DBEC scheme consists of invalidating and turning-off power to cache lines that are occupied by the "dead" instructions i.e., the instructions that are not "live" at a particular point of program execution. These are the instructions that would not be used again before being replaced in the cache. The information on whether an instruction is "dead" at a particular point of program execution is obtained from the compiler. The granularity at which the dead instructions are handled is a basic block. The compiler identifies basic blocks from the CFG and indicates the beginning and end of the basic blocks in the code. When the program is executed, these indications are used by the microarchitecture to turn-off dead blocks. The mechanism used, is explained below with an example.

A basic block [14], is a sequence of consecutive statements, in which the flow of control enters at the beginning and leaves at the end, without the possibility of branching, except at the end. Figure 1 shows a segment of a Control Flow Graph. B_is are the basic blocks. B_3 is a loop inside an outer loop containing basic blocks B_2, B_3 and B_4.

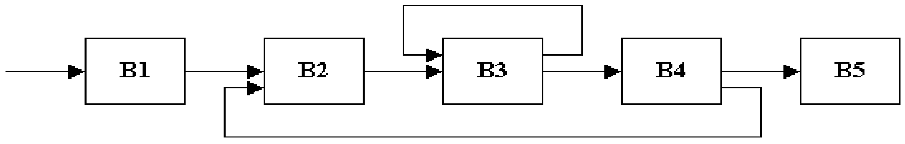

Fig. 1. Segment of a Control Flow Graph

The first statement of each basic block is annotated during the compilation stage. The annotations help to keep track of the basic blocks which are executed. B_2 is the block through which control enters the outer loop. The first statement of B_2 is annotated. Whereas B_3 and B_4 are part of the same outer loop and hence, the first statement of these blocks are not annotated. The key idea of DBEC is that the annotated statements help in identifying the blocks executed in the just completed outermost loop. This is done to make sure that B_2 and B_3 are not turned off when B_4 is under execution. When the annotated statement of B_2 is under execution, the block B_1 is completely executed. Hence, the cache lines completely occupied by the instructions of B_1 are turned-off. Similarly, the execution of first statement of B_5 will cause an invalidation of the lines occupied by the previous loop. i.e., as B_2, B_3 and B_4 are turned off. Thus, the instant at which the particular block is going to be "dead" is exactly determined, there is negligible performance degradation in this approach. The earlier approaches [4], [5], [6], have achieved power saving at the cost of performance. Further, this being a static approach supported by the compiler, the runtime architectural overhead of this approach is also negligible.

4 Hardware modification

The only hardware modification that is required in the DBEC approach is that, with each line of I-cache, one bit called the "turn-off bit" is added to the tag bits. These turn-off bits are initially unset at the start of program execution. The turn-off bits corresponding to lines of cache which contain the code for a loop are set. This is used to turn off these lines, when the execution of the loop is completed. This is achieved using the Gated-Vdd technique. The operation can be explained using the same example given in the previous section. Against execution of the first annotated statement of B_1, the processor sets the turn-off bits of those lines from which the instruction of B_1 are fetched. An instance of the various bytes in each line occupied by the B_1 block is shown in figure 2. Also, the turn-off bits for those lines after the execution of B_1 are shown in the

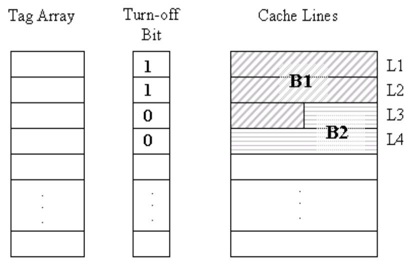

Fig. 2. A segment of the cache with tag bits

same figure. The turn-off bit of line L3 is not set, indicating that the line is to be retained even after the execution of B_1. This is because line L3 is only partially filled with code from B_1, and contains some code from B_2 also. When the control of execution reaches the beginning of block B_2, the annotated first statement of B_2 will cause the invalidation and turning-off of the lines whose turn-off bits were previously marked for this purpose. Hence, the lines L1 and L2 are now turned-off. The turn-off bit of line L3 is set only when B_2 is taken up for execution. When B_2 completes execution, line L3 is turned off.

5 Experimental methodology and Results

The SimpleScalar 2.0 toolkit [15] is used to implement the proposed idea. Benchmarks *art*, *equake*, *gzip* and *mcf* of the SPEC CPU 2000 benchmark [16] suite have been used to evaluate the performance.

5.1 Experimental setup

The out-of-order superscalar processor simulator of the SimpleScalar toolkit has been used to simulate switching off of the cache lines when annotated instructions are encountered in the instruction stream. All the benchmarks have been compiled using the C compiler gcc-2.6.3 for the SimpleScalar toolkit which has been modified to annotate instructions that aid switching off of cache lines.

The various phases in the compilation are as shown in figure 3. The source programs of the SPEC benchmark suite are compiled using *gcc* with the *-S* flag to get the assembly files. These assembly files are the input to the *annotator* which annotates instructions as explained below.

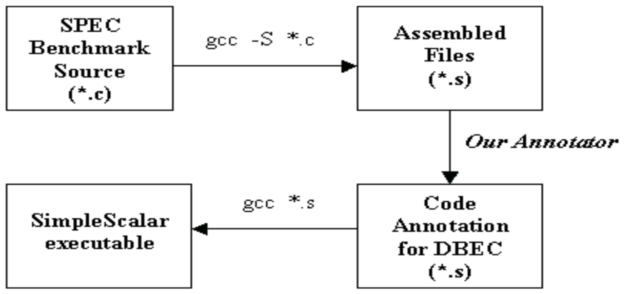

Fig. 3. Phases in the generation of annotated SimpleScalar executable

Parameter	Value
Fetch width	4 instructions per cycle
Decode width	4 instructions per cycle
Commit width	4 instructions per cycle
L1 I-cache	256, 512, 1024 and 2048 lines
L1 I-cache line size	32 bytes
L1 I-cache associativity	1-way (Direct mapped)
L1 I-cache latency	1 cycle
L1 D-cache	16 K, 4-way, 32 byte blocks
L2 unified cache	256 K, 4-way, 64 byte blocks
L2 unified cache latency	6 cycles

Table 1. Parameters used in the simulation

The *annotator* annotates instructions in three ways to aid the processor in switching off cache lines. The first type of annotation indicates that the cache lines whose "turn-off" bit has been set are to be switched off. This annotation is normally performed for instructions that begin a basic block. In the case of loops, only the first basic block of the loop is annotated so that the basic blocks within the loop do not switch off other basic blocks of the same loop. This ensures that cache lines containing the loop code are not switched off. The second type of annotation indicates that the next instruction is a "call instruction" within a loop. The third type of annotation indicates that the previous instruction was a function call which was part of a loop. The second and third annotations ensure that the functions which are called from within a loop do not switch off the cache lines that contain the code of that loop. The assembly files thus annotated are converted to SimpleScalar executable using *gcc*.

The implementation of the second and third type of annotation requires simple architectural support in the form of a dedicated counter. The annotation before the call increments the counter and the instruction following it decrements the same. Switching off of the cache lines is performed only when this counter's value is zero indicating that the code under execution is not part of any loop.

Program	Miss rate for the base model	Miss rate with DBEC
art	0.0001	0.0001
equake	0.0103	0.0103
gzip	0.02	0.02
mcf	0.0045	0.0047

Table 2. I-cache miss rates for the base model and the DBEC with 512 lines each

The instructions to manipulate the dedicated counter, which keeps track of entry and exit of calls, are added. An alternative is to have a dedicated register and use the increment and decrement instructions of the existing ISA before and after the function calls.

The SimpleScalar simulator is modified to switch off cache lines according to annotations added by the compiler. The simulator is used to collect results regarding power consumption.

5.2 Simulation Results

The main parameters considered in the evaluation of this implementation are the reduction in power consumption and miss rate. The leakage power for the cache is proportional to the total number of cache lines that are switched-on in the cache [9]. Hence the total number of lines switched off is taken to be an estimate of the power savings achieved.

$$\%\ \mathit{Power\ Saved}\ = \left(1 - \frac{\sum(\mathit{No.\ of\ active\ lines} \times \mathit{Duration\ of\ Activity})}{\mathit{Total\ number\ of\ lines} \times \mathit{Total\ duration}}\right) \times 100$$

Miss rate has been considered to study the impact of this technique on performance. The number of instructions considered while executing the benchmarks is 4 billion. The cache-related parameters used in the simulation model are given in table 1.

Table 2 gives the miss rates of the modified programs and the base programs of the SPEC 2000 benchmark suite. It can be seen that there is no significant difference in the miss rates. This is as expected, because the DBEC will turn-off a line only after an instruction is "dead". Thus the capacity and conflict misses of DBEC model will be same as the base model. Hence, there is no degradation in performance.

Figure 4 shows the power savings obtained for various benchmark programs as the number of cache lines varies from 256 to 2048, with all the cache lines initially on. It can be seen that, while three programs give a modest power saving of about 2-10%, one program *art*, records a maximum power saving of about 50% for 512 lines and about 40% for 1024 lines. This may be explained by the fact that DBEC consumes less power when the program consists of many independent loops, and *art* is one such program. The variation of the power saving for each program, as the number of lines of cache is varied shows an interesting pattern.

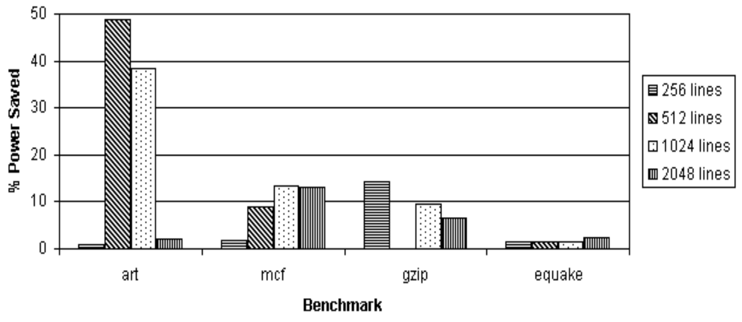

Fig. 4. Variation in I-cache power savings with number of cache lines

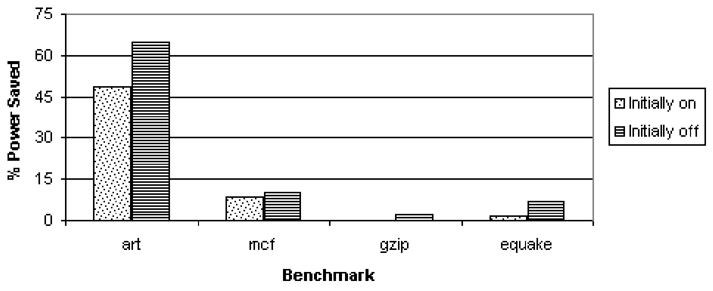

Fig. 5. Power savings for 512 line caches with lines initially switched on Vs off

It increases as the number of cache lines is increased from 256 onwards and then decreases as the number of cache lines is increased beyond 512/1024. One reason for this could be that, the cache lines are assumed to be initially on, and if all the lines of the cache are not used, the power saving decreases.

To study this further, the programs were executed with cache lines initially turned off. One of the programs (*art*), was run for different cache sizes. It was found that, as the number of cache lines increased from 256 to 2048, the power saving consistently increased. Figure 5 shows the comparison of power saved for a cache of 512 lines for the different programs. As expected, the power saving is higher when the cache lines are initially off.

Further, from figure 4 it is observed that the *equake* program shows a power saving of only 1.56% for a cache size of 512 lines. One reason for this could be that with 4 billion instructions *equake* would just be out of its initialisation phase [18]. To explore this further this program was run for 25 billion instructions and the fraction of the power saved was recorded at different instruction counts. The result of this is shown in figure 6. The power saved is not considerable till 7 billion instructions. Beyond 7 billion instructions, power saved gradually increases to reach a maximum of 23% at about 14 billion instructions and starts slowly decreasing. Actually, after 14 billion instructions the number of lines turned-off remain more or less the same but when averaged out over the total number of instructions, the value decreases.

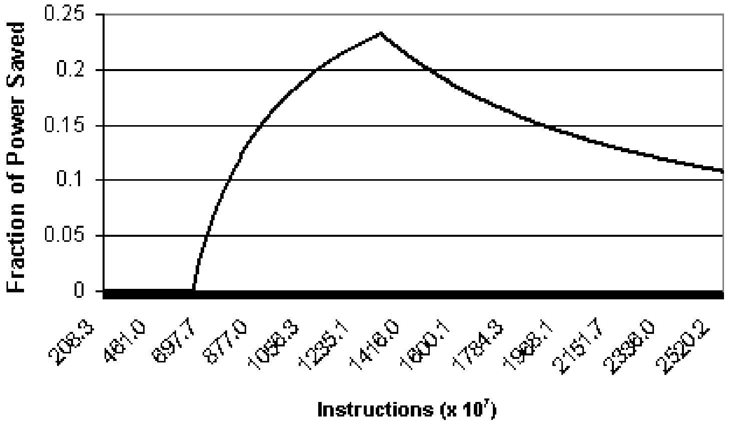

Fig. 6. Fraction of power saved versus instruction count for *equake*

6 Conclusion

The DBEC approach presented here, precisely identifies the blocks which are dead at a particular point of program execution with the help of the compiler. The performance degradation of this approach compared to the base model is almost negligible. Thus for this approach with very little performance degradation it is possible to get a power saving of about 15% and for certain programs a power saving of greater than 50%.

Some paths of the programs which are never executed (due to directed branches that are always taken in one direction) will not be turned off by this scheme. Further work has to be done in this direction. Moreover selective annotation can be used to get better saving at the cost of increased miss rate. The DBEC has been implemented as a direct mapped cache. This can be extended for associative caches with a little increase in hardware complexity.

References

1. Michael K Gowan, Larry L Biro and Daniel B Jackson: "Power Considerations in the Design of the Alpha 21264 Microprocessor," DAC98, San Francisco, CA, 1998, pp 726-731.

2. Trevor Mudge: “Power: A First class Design Constraint for Future Architectures,” IEEE Conference, HiPC, India, 2000, pp 215-224.
3. Nikolaos Bellas, Ibrahim Hajj, Constantine Polychronopoulos and George Stamoulis: “Architectural and Compiler support for Energy Reduction in the Memory Hierarchy of High Performance Microprocessors,” ISLPED, ACM Press New York, USA, 1998, pp 70-75.
4. Michael D Powell, Se-Hyun Yang Babak Falsafi, Kaushik Roy and T N Vijayakumar: “Gated-Vdd: A circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories,” ISLPED, 2000, pp 90-95.
5. Se-Hyun Yang, Michael D Powell, Babak Falsafi, Kaushik Roy and T N Vijayakumar: “An Integrated Circuit/Architectural Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches,” Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), Jan 2001.
6. Huiyang Zhou, Mark C Toburen, Eric Rottenberg and Thomas M Conte: “Adaptive Mode-Control: A Low-Leakage, Power-Efficient Cache Design,” TR, Dept. of Electrical & Computer Engg. North Carolina State University, Raleigh, NC, 27695-7914, Nov 2000.
7. Kanad Ghose and Milind B Kamble: “Reducing Power in Superscalar Processor Caches Using Subbanking, Multiple Line Buffers and Bit- Line Segmentation,” ISLPED, ACM Press, New York, USA, 1999, pp 70-75.
8. Hongbo Yang, Guang R Gao, Andres Marquez, George Cai and Ziang Hu: “Power and Energy Impact by Loop Transformations,” `http://research.ac.upc.es/-pact01/colp/paper12.pdf`
9. Stefanos Kaxiras, Zhigang Hu, Girija Narlikar and Rae McLellan: “Cache-Line Decay: A Mechanism to Reduce Cache Leakage Power,” IEEE workshop on Power Aware Computer Systems (PACS), Cambridge, MA, USA, 2000, pp 82-96.
10. Victor Delaluz, Mahmut Kandemir, N Vijayakrishna, Anand Sivasubramanian and Mary Jane Irwin: “Hardware and Software Techniques for Controlling DRAM Power Modes,” IEEE Trans. on Computers, Vol.50, No11, Nov 2001, pp 1154-1173.
11. Krishna V Palem, Rodric M Rabbah, Vincent J Mooney III, Pinar Kormatz and Kiran Puttaswamy: “Design Space Optimization of Embedded Memory Systems via Data Remapping,” CREST-TR-02-003 GIT-CC-02-011, Feb 2002.
12. John Hennessy: “The Future of Systems Research,” IEEE Computers, Aug 1999, pp 27-33.
13. J.Kin, M.Gupta and W.Mangione-Smith: “The Filter Cache: An Energy Efficient Memory Structure,” Proc. IEEE Int’l Symp. Microarchitecture, IEEE CS Press, 1997, pp 184-193.
14. Alfred V Aho, Ravi Sethi and Jeffrey D Ulman: “Compilers: Principles, Techniques and Tools,” Addison-Wesley, ISBN : 817-808-046-X, Third Indian Reprint 2000.
15. D Burger, Todd M Austin: “The Simplescalar Tool Set, version 2.0 :,” CSD Technical Report #1342. University of Wisconsin-Madison, June 1997.
16. “SPEC CPU 2000 benchmark suite,” `http://www.spec.org`
17. Stefanos Kaxiras, Zhigang Hu and Margaret Martonosi: “Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power,” Proc. of Int’l Symp. Computer Architecture, ISCA, ACM Press, New York, USA, 2001, pp 240-251.
18. Suleyman Sair and Mark Charney: “Memory Behavior of the SPEC2000 Benchmark Suite,” Technical Report, IBM, 2000.