

# Compiler-Directed Data Cache Leakage Reduction

Wei Zhang

Department of ECE, Southern Illinois University Carbondale

## Abstract

*Leakage energy reduction for caches has been the target of many recent research efforts. In this work, we propose a novel compiler directed approach to reduce the data cache leakage energy by exploiting the program behavior. The proposed approach is based on the observation that only a small portion of data are active at runtime and the program spends a lot of execution time in loops, so a large portion of data cache lines, which are not accessed by the loops, can be placed into leakage control mode to reduce leakage energy consumption. The experimental results show that the proposal approach is comparable to the pure hardware based approach in reducing data cache leakage energy.*

## 1. Introduction

Cache leakage energy reduction has been the target of many recent research efforts. While most studies focus on reducing cache leakage energy by using circuit or architectural techniques [3, 4, 1, 5], a compiler-directed leakage energy optimization strategy is proposed for instruction cache [2]. For the data cache, Zhang [9] et al. recently present code restructuring techniques for array-based and pointer-intensive applications to reduce leakage energy. While this approach can reduce the data cache energy consumption significantly, it may increase the code size and thus impact performance by inserting activate/deactivate instructions at the cache line granularity. In contrast, in this paper, we propose a compiler-directed data cache leakage optimization strategy at the loop granularity (also called loop-aware approach in this paper).

The proposed approach is based on the observation that only a small portion of data are active at runtime and the program spends a lot of execution time in loops, so a large portion of data cache lines, which are not accessed by the loops, can be placed into leakage control mode to reduce leakage energy consumption. In this work, we propose an optimistic approach to place the cache lines into low leakage mode during the execution of innermost loops.

The rest of the paper is organized as follows. Section 2

presents the compiler-directed data cache leakage optimization strategy. Section 3 shows the experimental results. Finally, we draw our conclusions in section 4.

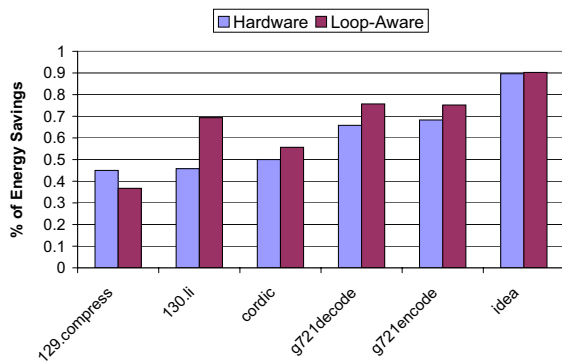
## 2. Compiler-Directed Data Cache Leakage Reduction

It is well known that in general, a program spends a large percentage of time executing only a small portion of the program, i.e., loops, which have been the targets of traditional performance-oriented optimizations. If the size of the data accessed by the loop is less than the size of the data cache, the rest of the cache lines can be placed into low leakage mode during the loop execution. And since loops are likely to take a long period of time to execute, this strategy can result in significant leakage energy savings.

While theoretically, the compiler can choose arbitrary regions of code and apply the proposed strategy to reduce the data cache leakage energy, the exploration of the large design space is beyond this paper. In this work, the proposed strategy works on the innermost loop granularity to reduce leakage energy, since the innermost loop provides a good tradeoff between large execution time and small data footprint. We assume the support of `turn_on/turn_off` instructions to manage the leakage mode of L1 data cache. Thus the compiler's job is to insert an `turn_on` instruction before an innermost loop and an `turn_off` instruction after an innermost loop. We also implement an optimization phase to remove redundant `turn_on/turn_off` instructions for adjacent loop regions. To reduce the performance penalty of activating each data cache line that is accessed by the streamlining code, we employ the just-in-time activation technique proposed in [10]. For loops that access a large set of data (close to or larger than the size of the data cache), we propose to use loop transformations (i.e., loop tiling and loop distribution) to reduce the data footprints of innermost loops before applying the proposed approach.

## 3. Experiment

We employ the state-preserving leakage control mechanism introduced in [1]. The proposed compiler-directed



**Figure 1. Percentage of leakage energy savings**

strategy is implemented in trimaran 2.0 [6]. The VLIW configuration used in our experiments has four IALUs (integer ALUs), two FPALUs (floating-point ALUs), one LD/ST (load/store) unit and one branch unit. The energy numbers are the same as those in [2]. To ensure diversity, we use a suite of six benchmarks from SPEC INT 95 [7] (129.compress, 130.li) and Mediabench [8] (cordic, g721decode, g721encode and idea).

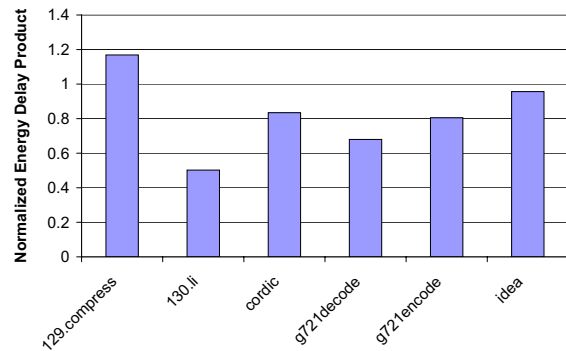
### 3.1. Energy and Performance Results

Figure 1 shows the percentage of leakage energy savings for the loop-aware approach and the pure hardware based approach, which uses a 2-bit saturating counter to manage the leakage of each cache line [4]. Except 129.compress, we find that the proposed strategy leads to more leakage energy reduction. In average, the loop-aware approach can save 6.4% more leakage energy of L1 data cache than the pure hardware based approach.

The energy delay product results of the loop-aware approach are presented in figure 2. To compare our approach with the pure hardware based approach, the results are normalized with the energy delay product of pure hardware based approach. We find that except 129.compress, the loop-aware approach has a better energy delay product than the pure hardware based approach (the less the better). These results show that the loop-aware approach is very competitive in terms of energy and energy delay product, compared to the pure hardware based approach.

## 4. Conclusions

In this paper, we propose a novel compiler directed approach to reduce the data cache leakage energy. Our strat-



**Figure 2. Normalized energy delay product for loop-aware approach.**

egy is based on the observation that if program hotspots only access a limited number of data, the rest of the “cold” cache lines can be placed into low leakage mode during loop execution. The experimental results show that the loop aware approach is comparable to the recently proposed pure hardware based approach in reducing data cache leakage energy.

## References

- [1] K. Flautner et al. Drowsy caches: Simple techniques for reducing leakage power. ISCA 2002.
- [2] W. Zhang et al. Compiler-directed instruction cache leakage optimization. MICRO, 2002.
- [3] M.D. Powell et al. Reducing leakage in a high-performance deep-submicron instruction cache. IEEE Transaction on VLSI, Vol. 9, No. 1, February 2001.
- [4] S. Kaxiras, Z. Hu, M. Martonosi. Cache decay: exploiting generational behavior to reduce cache leakage power. ISCA 2001.
- [5] S. Heo, K. Barr, M. Hampton, and K. Asanovic. Dynamic fine-grain leakage reduction using leakage-biased bitlines. ISCA 2002.
- [6] <http://www.trimaran.org>.
- [7] <http://www.spec.org>.
- [8] C. Lee et al. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. MICRO 1997.
- [9] W. Zhang et al. A compiler approach for reducing data cache energy. ICS, 2003.
- [10] J. S Hu et al. Exploiting program hotspots and code sequentiality for instruction cache leakage management. ISLPED 2003.