Energy–Efficient Design for Highly Associative Instruction Caches in Next–Generation Embedded Processors

Juan Luis Aragon   Dan Nicolaescu   Alex Veidenbaum   Ana–Maria Badulescu

Center for Embedded Computer Systems
University of California at Irvine
{jlaragon, dann, alexv}@cecs.uci.edu

Abstract

This paper proposes a low-energy solution for CAM-based highly associative I-caches using a segmented wordline and a predictor-based instruction fetch mechanism. Not all instructions in a given I-cache fetch are used due to branches. The proposed predictor determines which instructions in a cache access will be used and does not fetch any other instructions. Results show an average I-cache energy savings of 44% over the baseline case and 6% over the segmented case with no negative impact on performance.

1. Introduction

Embedded processors routinely use multiple instruction issue to increase performance. This leads to higher energy consumption due to the presence of additional resources and higher utilization of such resources. These processors use in-order instruction issue and often do not contain a floating point unit. As a result, I- and D-caches and TLBs consume a major share of overall energy (from 10% to 25% was reported in [6]).

Today such a processor typically fetches two instructions per cycle from an I-cache which needs a wide data store, increasing its energy consumption. I-caches are also highly associative: 32-way set associative for Intel StrongArm [6] and Xscale [2], 16-way for Transmeta Crusoe [5]. High associativity implementations using CAM and SRAM arrays use even more energy.

The goal of this research is to propose a reduced energy I-cache design for future embedded processors. It assumes that higher clock frequencies, longer pipelines, wider issue and highly associative CAM–based caches will be used in such processors, and explores opportunities to reduce I-cache energy consumption.

This paper attempts to reduce the instruction fetch energy in two ways. First, it proposes a modification to the data array organization to use a “segmented” wordline. This is similar to proposals for using a segmented bitline [3] and saves energy by allowing the fetch of the exact number of instructions needed in a cycle as determined by issue width N. Second, it further reduces the number of instructions read by fetching only the “useful” ones among the N-word fetch segment. When a processor fetches N instructions per cycle, not all may actually be issued. This happens in two cases:

1. One of the N instructions is a conditional or an unconditional branch that is taken – a *branch out* case. All instructions after the taken branch are unused.
2. An I-cache line contains a branch target, which is not at the beginning of the N–word segment – a *branch in* case. The instructions before the target are unused.

A predictor is proposed to identify which instructions in each fetched line are going to be used. Based on this prediction only the useful part of the cache line is fetched in each clock cycle.

There are many hardware and architectural proposals for reducing the energy consumption in I-caches but most of them are not applicable in the case of a highly-associative CAM-based design. Two very related approaches are a banked cache organization and a segmented wordline or bitline RAM design.

A banked I-cache organization divides the cache into sub-banks [3] and activates only the required sub-banks. A segmented wordline [7] is used to reduce the length of a wordline and thus its capacitance. Bitline segmentation [7, 3] divides a bitline using pass transistors and allows sensing of only one of the segments.

2. Fetch Mask Predictor

The *Fetch Mask Predictor* determines the control bit vector used to decide which words in a instruction cache line will be fetched each cycle. Each bit in the mask controls the drivers in the segmented wordline in order to access only the useful words in the next fetch cycle.

For *branching into* the next line, it is only necessary to determine whether the current fetching line contains a taken branch. This is provided by both the Branch Target Buffer (BTB) and the branch predictor. When a branch is taken and the target address is known, its position in the next I-cache

1530-1591/04 $20.00 (c) 2004 IEEE

line is easily determined. The corresponding *target mask* will select those instructions from the target until the end of the line.

For *branching out* of the next line, it is necessary to predict if the next I-cache line contains a branch that is going to be taken. In that case, the instructions from the branch position to the end of the line do not need to be fetched in the next cycle. This is accomplished by a *Predicted Mask Table* (PMT) which has the same number of entries as the I-cache. Each PMT entry stores whether the corresponding I-cache line contains a branch that will be taken the next time the branch is predicted.

It is possible for both *branching in* and *branching out* to occur on the same cache line. In this case, the *target mask* and the *predicted mask* need to be combined. There is no performance degradation associated with the use of the *Fetch Mask Predictor* since it basically anticipates the behavior of the underlying branch predictor mechanism.

3. Experimental Evaluation

The Wattch v1.02 power simulator [1] was augmented with the power model implemented in CACTI v3.2 [7] in order to increase its accuracy. The embedded processor modeled is a 4-wide issue processor with a pipeline of 12 stages. The CMOS process parameters are a 1.5GHz clock and a $.10\mu$m feature size. The cache organization was based on the Xscale processor: 32KB, 32–way set associative data and instruction L1 caches with 32 byte lines. There was no L2 cache. Benchmarks from MiBench [4] were used, compiled with the -O3 flag and run to completion using the "large" input set.

The baseline I-cache operation is as follows. Every cycle a new I-cache access is performed to fetch N instruction words. An active CAM match line selects a data array line and all 256 bits (8 instructions) are read out. A multiplexor is used to select an aligned set of N instruction words to be sent to the instruction decoder. On the other hand, the segmented wordline design allows partial accesses to the I-cache lines. Finally, the *Fetch Mask Predictor* provides the control bit mask to enable access to a subset of the N words in the data array as explained above. All other words in the data array are disabled. The average I-cache energy savings for each benchmark category are shown in Figure 1. On average, the segmented wordline design saves approximately 39% of the I–cache energy over the baseline design. The fact that half of the cache line is disabled in this case, and that the I-cache consumes more than one half of the dynamic energy justifies the above results.

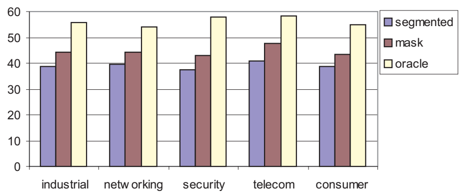

Figure 1. Average I–cache energy savings

The mask predictor performs better, saving on average an additional 6% of the I-cache energy over the segmented design. In order to asses the potential of the proposed prediction technique an oracle predictor was used. It makes no mistakes in identifying the instructions to be fetched. The oracle allows an additional 12% of I-cache energy reduction. This indicates that the mask predictor design can be significantly improved. Finally, Figure 2 shows the overall processor energy savings. They range from 25% to 31% for the segmented wordline design, and from 29% to 37% for the *Fetch Mask Predictor* design.

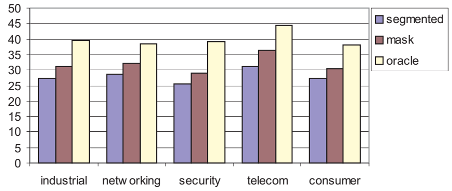

Figure 2. Overall processor energy savings

4. Conclusions

Two energy saving mechanisms utilizing a segmented wordline for a highly associative CAM-based cache design have been presented. They were shown to reduce the energy consumption of the I-cache by 44% for a 4-issue next generation embedded processor. It was also shown that the *Fetch Mask Predictor* approach has potential for significant improvement. Future embedded processor are likely to have even larger caches, branch predictors and BTBs. This will further improve the efficiency of the *Fetch Mask Predictor*.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA–27 Proceedings*, pages 83–94, 2000.

[2] L. T. Clark and et al. An embedded 32b microprocessor core for low-power and high-performance applications. *IEEE JSSC*, 36(11):1599–1608, Nov. 2001.

[3] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *Proceedings ISLPED*, pages 70–75. ACM Press, 1999.

[4] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, pages 83–94, 2001.

[5] A. Klaiber. The technology behind Crusoe processors. Technical report, Transmeta Corporation, Jan. 2000.

[6] J. Montagnaro and et al. A 160–MHz, 32–b, 0.5–W CMOS RISC Microprocessor. *IEEE JSSC*, 31(11):1703–1714, Nov. 1996.

[7] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power, and area model. Technical report, Digital Equipment Corporation, 1990.