Predictive Line Buffer: A fast, Energy Efficient Cache Architecture

Kashif Ali MoKhtar Aboelaze SupraKash Datta
Department of Computer Science and Engineering
York University
Toronto ON CANADA

Abstract

Two of the most important factors in the design of any processor are speed and energy consumption. In this paper, we propose a new cache architecture that results in a faster memory access and lower energy consumption. Our proposed architecture does not require any changes to the processor architecture, it only assume the existence of a BTB. Using Mediabench, a benchmark used for embedded applications, Simplescalar simulator, and CACTI power simulator,we show that our proposed architecture consumes less energy, and have better memory access time, than many existing cache architectures.

1 Introduction

The processor speed is advancing at a much higher rate than the memory access speed. The cache memory is generally considered to be the only solution to bridge the gap between the processor speed and the memory speed. By placing a small cache memory between the CPU and the main memory, memory access could be reduced to one or two cycles. Another reason for the popularity of the cache memory, is that cache access requires less energy than accessing the main memory. With the increasing demands of portable (battery operated) devices, energy consumption has become a very important factor in the design of the CPU.

In today's processors, cache memory takes a very large portion of the area of the CPU, even a larger portion of the transistor count. A large

cache requires longer access time, and consumes more energy due to the increase in the bit-line and word-line capacitance [10]. Adding more cache to improve performance is not the ideal solution in this case. But rather a good cache architecture that can produce a good performance in terms of average memory access time, and energy consuption.

In this paper, we propose a new cache architecture that consists of the cache, and a single line buffer between the cache and the CPU. Adding a line buffer is not a new concept. However, the main contribution of the paper is the prediction mechanism used where the CPU predicts if the next access is from the line buffer, or the main cache. Thus, saving at least one cycle when we miss on the line buffer. Using Simplescalar and CACTI power simulator, and Mediabench testbench we show that our proposed architecture have a faster memory access, and lower energy consumption compared to single line buffer without prediction, filter cache, and hotspot cache.

The organization of the paper is as follows. In section 2 we present some recent attempts to reduce energy and memory access time. Section 3 introduces our proposed architecture. Section 4 explains the simulation setup, and the simulation results and compares it with the above mentioned architectures. Section 5 is a conclusion and future work.

2 Previous Works

There have been many attempts to improve the cache performance (average memory access time) as well as decreasing the energy consumption of the cache. In this section, we briefly discuss some of recent attempts of reducing energy consumption, and the average memory access time.

Jouppi in [4] showed how to use a small fully associative cache and prefetching to improve the performance of a direct-mapped cache. The authors in [9] showed how to tune the filter cache to the needs of a particular application in order to save energy.

Hotspot cache was introduced in [11] where frequently executed loops are detected and placed in a small filter cache. Their architecture led to a less energy consumption than a regular filter cache. In [1] the authors proposed a variable sized block cache. Their scheme depends on identifying the basic blocks (block tail is a backward branch, block head is the target of the backward branch) and they mapped them to a variable size cache block. They successfully addressed the problem of the instruction overlap among traces

that was present in the trace cache [6]. In [5] the authors investigated the energy dissipation in the bit array and the memory peripheral interface circuits. Taking these parameters into consideration, they optimized the performance and power dissipation in the cache. Different techniques for reducing static energy consumption in multiprocessors caches were compared in [3]. While [2] proposed code compression techniques, where the instructions are accessed from the cache in a compressed form.

3 Proposed Architecture

In this paper, we assume a cache memory with a single line buffer. We also propose a mechanism to predict if the next access is from the line buffer or from the cache. If our prediction is correct, then we save one clock cycle (the miss in the line buffer) together with the energy consumed in accessing the line buffer.

3.1 Conventional line buffer

In a conventional line buffer a cache line is placed in front of L1 cache to capture the spatial locality of reference. Once a word is accessed , the line containing that word is transfered to the line buffer. The next access, if sequential, will be accessed from the line buffer instead of the cache. The energy required to access the line buffer is much less than the energy required to access the cache. In case of a miss from cache line, it requires an extra clock cycle to fetch the instruction from L1. If we can access the cache in the same cycle as the line buffer, that saves the extra cycle, but increases the cycle time.

3.2 Predictive Line Buffer

To avoid performance overhead of using conventional line buffer, we now propose a new scheme. The key difference is that our scheme use prediction between line buffer and L1 cache. Predictive line buffer architecture is shown in Fig. 1. By dynamic steering between line buffer and L1 cache we avoid the extra cycle therefore improving both energy and average access time.

Our technique does not add any extra hardware to the architecture except the prediction mechanism. Also it assumes the existence of a BTB which is

common in almost every modern processor. The prediction mechanism itself consists of few multiplexers and comparators.

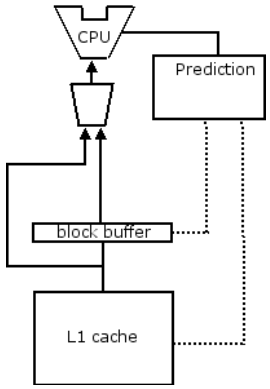

Figure 1: Cache with Line Buffer and Predictive Steering Scheme

3.3 Implementation Details

The main idea in our prediction is the following. The access of the next instruction is from the line buffer except in 2 cases.

1. If the current instruction is a control transfer instruction, and will be taken. Or

2. If the current instruction is not a control transfer, but the address of the current instruction at the end of the line buffer.

During the fetch stage, the CPU accesses the BTB in order to now if the current instruction is a control transfer or not, and if a control transfer, where is the target address if taken. The decision is sent to he control unit. Also the control units checks the least significant $\log k$ bits of the instruction address where k is the size of the line buffer, if they all ones, the access is from the cache, otherwise the access is from the blockbufer. Figiuref 2 shows the pseudocode for the prediction mechanism. LAST-ADDR means that the last accessed byte is the last byte in the line buffer . The exact details of how to calculate that condition depends on the particular memory system. for a byte addressable word referenced processor with cache line of k bytes, the condition is (PC/4) mod k=k-1.

```
/*Fetch modes:
* Modes_L1: fetch from the cache
Mode_B1: fetch from the line buffer
*
*Instruction types:
INST_UNCTRL: non control transfer instruction
INST_CTRL: a control transfer instruction
*/
/* Initialization */
fetch_mode := MODE_L1
inst_offset := offset(PC)
inst_type := OPCODE(PC)

if(inst_type == INST_CTRL  and predicted taken) {
                fetch_mode := MODE_L1
                }
                else if (PC == LAST_ADDR) {
                fetch_mode := Mode_L1
                }
                else {
                fetch_mode := MODE_L1
                }
```

Figure 2: Pseudocode for Single Predictive Line Buffer

4 Experimental Results

In this section, we'll compare energy and avenge memory access time of predictive line buffer with conventional line buffer, Filter Cache, and HotSpot Cache using Direct Mapped Caches as our base model.

4.1 Experiment Setup

We use SimpleScalar toolset [7] along with CACTI [8] to conduct our experiments. We have modified SimpleScalar to simulate Filter Cache, line buffer, hotspot Cache and Predictive line buffer. Our base architecture is using 16KB direct mapped cache with 32 bytes line size. our line buffer is also of 32 bytes line size. We also assumed 512 bytes, direct-mapped L0 cache for both Filter and HotSpot Cache. The BTB has 4-way set-associative with 64 sets. We also assumed 2-level branch predictor in our simulation. We evaluated energy consumption using $0.35\mu m$ process technology. Table 1 shows the energy consumption per-access for line buffer, L0 and L1 cache. For HotSpot cache, we used a threshold value of 16 as suggested in [11]. We used Mediabench with each applications executing 500 Million instructions.

Table 1: Energy consumed per access

Cache	Energy
Line Buffer	0.12nJ
L0 Cache	0.69nJ
L1 Direct-Map	1.63nJ
L1 Set-assoc	2.49nJ

4.2 Energy

Fig 3 shows the normalized energy consumption of the predictive line buffer, regular line buffer, filter cache and Hotspot cache, normalized to a directmapped cache for the different programs in the mediabench testbench. While Table 2 compares the average energy consumption (over the Mediabench suite) of the same cache architectures. It is obvious that our prosed architecture consumes less energy than the other three architectures.

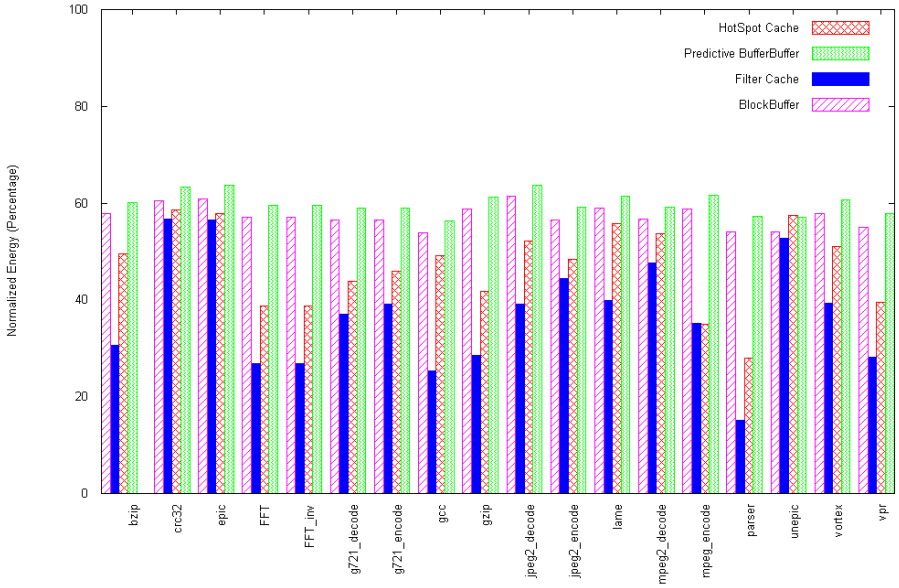

Figure 3: Normalized energy consumption of Filter Cache, conventional line buffer predictive line buffer and hotSpot Cache using direct-map as L1 cache

4.3 Average memory Access Time

The average memory access time of the predictive line buffer cache for the Mediabench suite is shown in Figure 4. While Table 2 compares the average memory access time for our proposed architecture and compares it with the three mentioned architecture. Our architecture has almost an average memory access of 1.

5 Conclusion

We proposed Instruction cache architectural architecture that has a low energy consumption and a fast average memory access time. Depending on the current instruction we set the next instruction be fetched from either the line buffer or L1 cache. Our scheme uses existing hardware and does not require any modification to the CPU architecture. Predictive line buffer scheme could be used with direct-map or set-associative cache. Our architecture reduces the instruction caches energy by up to 64%.

Table 2: Various Schemes Average Energy Reduction Using Direct-map

Scheme	Energy consumption	Average memory access time
Line buffer	57.33	1.290
Filter Cache	42.20	1.21
HotSpot Cache	46.94	1.066
Predictive Line buffer	59.98	1.004

References

[1] Beg, A. Yul C. "Improved instruction fetching with a new block-based cache scheme" *Proc. of the International Symposium on Signals, Circuits and systems ISSCS2005* Vol. 2, pp 765-768 14-15 July 2005

[2] Benini, L; Macii, A.; Nannarelli, A.; "Code compression architecture for cache energy minimization in embedded systems" *IEE Proceedings on Computers and Digital Techniques* Vol. 149, No. 4. pp 157-163 July 2002.

[3] Hanson, H.; Hrishikesh, M.S.; Agarwal, V.; Keckler, S.W¿; Burger, D. "Static energy reduction techniques for multiprocessor caches" *IEEE Transactions on Very Large Scale Integration Systems* Vol. 11, No. 3 pp 303:313 June 2003.

[4] Jouppi, N.P. "Improving direct-mapped cache performance by the addition of a small fully associative cache and prefetch buffers" *the* 17^th *Annual International Symposium on Computer Architecture ISCA* pp 364:373 May 1990.

[5] Ko, U. Balsara, P.T.; Nanda A.K. "Energy optimization of multilevel cache architecture for RISC and CISC processors" *IEEE Transactions on Very Large Scale Integration* Vol. 6, No. 2 pp 299:308 June 1998

[6] Rotenberg E. et al "A trace cache microarchitecture and evaluation" *IEEE Transactions on Computers* Vol. 48, No. 2 pp 111-120 Feb. 1999.

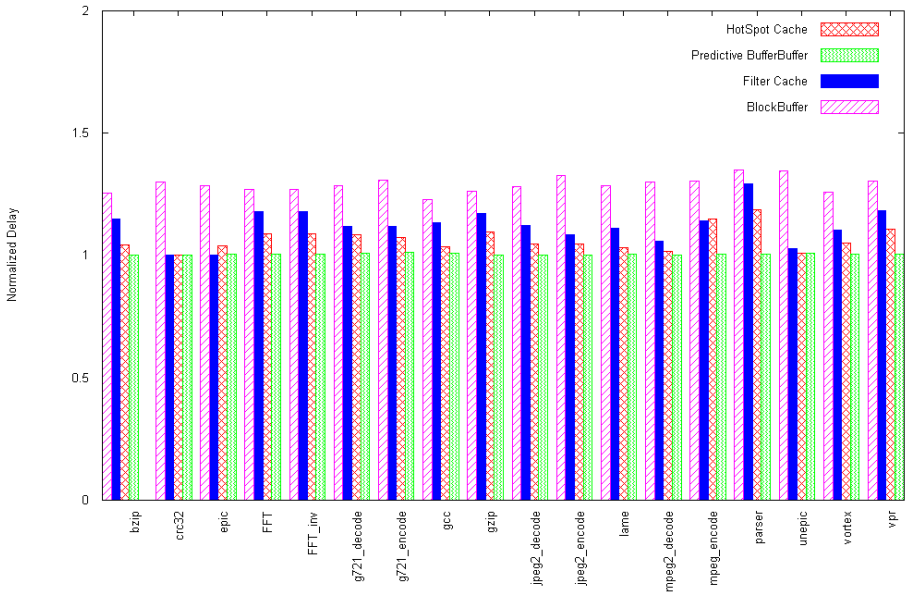

Figure 4: Normalized delay for Filter Cache, conventional line buffer, predictive line buffer and hotSpot Cache using direct-map L1 cache

[7] The Simplescalar simulator *www.simplescalar.com* 2005

[8] Shivakumar, P.; Jouppi, N.; "CACTI 3.0: An integrated cache timing, power, and area model" *Technical Report 2001.2 Compaq Research Lab* 2001

[9] Vivekanandarajah, K.; Srikanthan, T.; "Custom instruction filter cache synthesis for low-power embedded systems" *The 16th IEEE International Workshop on Rapid System Prototyping, (RSP 2005).* pp 151-157 June 2005.

[10] Weste, N. and Eshraghian K. *Principles of CMOS VLSI Design* Addison-Wesley N.Y. 1993

[11] Yang, C.-L.; Lee, C.-H.; "HotSpot cache: joint temporal and spatial locality exploitation for I-cache energy reduction" *Proc. of the 2004 International Symposium on Low Power Electronics and Design ISPLD'04* pp 114:119 Aug. 2004.