PAPER

An Accurate and Energy-Efficient Way Determination Technique for Instruction Caches by Using Early Tag Matching

Eui-Young Chung[†], Cheol Hong Kim[††], *and* Sung Woo Chung[†††], *IEEE Members*

SUMMARY Energy consumption has become an important design consideration in modern processors. Therefore, microarchitects should consider energy consumption, together with performance, when designing the cache architecture, since it is a major power consumer in a processor. This paper proposes an accurate and energy-efficient way determination (instead of prediction) technique for reducing energy consumption in the instruction cache by using early tag matching. Way prediction has been considered as one of the most efficient techniques to reduce energy consumption in the caches. The proposed scheme allows early tag matching for accurate way determination. With this feature, our scheme drastically improves the way determination accuracy compared to the previous way prediction techniques. To enable the early tag matching, the tag lookup stage is inserted prior to the fetch stage in the pipeline architecture. The tag matching is performed during the tag lookup stage, and then only one way is accessed during the fetch stage, leading to good energy efficiency. Simulation results show that the proposed technique reduces the energy consumption in the instruction cache by 55.1% on average. Moreover, our technique guarantees negligible performance degradation by overlapping two pipeline stages in case of branch misprediction.

key words: *Low power design, Instruction Cache, Way prediction, Tag matching*

1. Introduction

In modern processors, a significant fraction of total energy is consumed in instruction fetches. Brooks et al. found that instruction fetch accounts for 22.2% of the energy consumed in the Intel Pentium Pro processor [4]. Most of the energy for instruction fetch is consumed by the instruction cache. For example, the instruction cache is responsible for 27% of the energy consumed in the whole SA-110 [12]. For this reason, energy as well as performance should be simultaneously considered for instruction fetch design.

A number of architecture-level approaches have been introduced to improve the energy efficiency of the instruction cache. Bellas et al. proposed a technique that uses an additional mini cache located between the instruction cache and the CPU core in order to reduce signal switching activities and dissipated energy with the help of compilers [3]. Selective-way cache provides the ability to disable a set of the ways in a set-associative cache during the periods of modest cache activities to reduce energy consumption, whereas the full cache may remain operational for more cache-intensive periods [1]. Partitioned instruction cache reduces the size of cache to be activated by using a prediction technique, leading to good energy efficiency [10][11].

Modern microprocessors generally employ set-associative instruction caches to achieve low miss rates by reducing conflict misses. However, in the perspective of power consumption, set associative caches are inferior to direct-mapped caches, since all the ways in a set should be activated in parallel when the cache is accessed. For example, 4-way set-associative caches precharge and read four ways but select only one of them on a cache hit, wasting energy for three irrelevant ways. However, for better performance, set-associative caches are popularly used not only in high-performance processors but also in embedded processors such as ARM-family processors [15].

To improve the energy efficiency of set-associative caches, way prediction techniques have been proposed [7][8][13][14]. However, previous way prediction techniques show performance loss or little energy reduction due to the low prediction accuracy. In this paper, we propose a simple and accurate way determination technique by inserting an extra pipeline (stage tag lookup stage for way determination) prior to the fetch stage. In the conventional pipeline architecture, the branch prediction is performed prior to the fetch stage. In our scheme, the tag lookup stage is inserted between the branch prediction stage and the fetch stage for the way determination. The inserted tag lookup stage does not require additional hardware logic, but it is moved from the fetch stage.

In the previous way prediction techniques, the tag matching and instruction fetch are performed in parallel during the fetch stage. However, the proposed technique divides the fetch stage into two consecutive stages (tag lookup and fetch stage). The first stage (tag lookup stage) performs only tag matching for way determination, and the second stage (fetch stage) performs the instruction fetch. Therefore, the tag matching and the instruction fetch are performed sequentially in the proposed architecture. This seems to be similar to the serial cache[14]. However, the performance loss is much less, since the proposed technique can hide addi-

[†]School of Electrical and Electronic Engineering, Yonsei University, Seoul 120-749, Korea

[††]School of Electronics and Computer Engineering, Chonnam National University, Gwangju, 500-757 Korea

[†††]Division of Computer and Communication Engineering, Korea University, Seoul 136-713, Korea, Corresponding author, E-mail: swchung@korea.ac.kr

tional branch misprediction penalties from the inserted tag lookup stage except the case when the branch target is mispredicted by BTB (Branch Target Buffer). (Please note that branch prediction penalty is proportional to the pipeline length. Details are explained in Section 3.2.) The proposed technique reduces the energy consumption in the instruction cache significantly by accessing only a single way in the cache thanks to early tag matching.

The rest of this paper is organized as follows. Section 2 presents related research. Section 3 describes the proposed way determination technique and shows the differences from the Serial Cache. Section 4 discusses our evaluation methodology and shows detailed evaluation results followed by the conclusions in Section 5.

2. Background

Caches consume significant portion of total processor energy consumption, and the cache access is known as one of the most timing-critical paths in most processors. For this reason, cache architecture is very important when designing processor architecture. According to the results obtained from CACTI model [16], shown in Table 1, the normalized per-access energy consumption of the cache increases as the degree of associativity increases. Moreover, a direct-mapped cache shows the best access latency. However, the direct-mapped cache shows the worst hit rates due to frequent conflict misses. A fully-associative cache provides the highest hit rates, but shows the worst energy efficiency and access latency.

The alternative is a set-associative cache for tradeoff between hit rates and access latency, which is used in most processor architectures. In a set-associative cache, multiple cache lines with same index in a set are accessed simultaneously but at most one of them is fetched, resulting in unnessary energy consumption. To reduce the dynamic energy consumed by multiple cache line accesses, way prediction techniques have been proposed. Way predicting set-associative caches initially access a single tag and its corresponding data array based on their prediction mechanism, and access the other arrays only when the initial access does not result in a match, which leads to less energy consumption at the expense of longer access time in case of a way misprediction.

In the past, processor designers hesitated to adopt the way prediction schemes due to their low prediction accuracy, resulting in significant performance degradation [9]. To overcome this weakness, there have been many studies on the way prediction to improve the prediction accuracy. Inoue et al. adopted an MRU algorithm for way prediction and showed reduced cache access time by improving the prediction accuracy [7][8]. Powell et al. examined the impact of the way prediction on the energy consumption of the instruction cache [13]. For the precise way prediction, they associated a way prediction with the PC (Program Counter) of the previous instruction address to utilize the relation between the branch predictor and the instruction cache. However, the way prediction is performed based on the lastly accessed history in their technique, resulting in a strong dependency on the access history. Reinman et al. compared both performance and energy efficiency of a serial cache, where the tag matching and the access to the data are performed sequentially, to those of a cache with the way predictor [14]. The performance of the serial cache and that of the cache with the way predictor show little difference in terms of performance. Moreover, the serial cache requires low hardware complexity compared to the cache with a way predictor. However, their architectural parameters in the simulations were quite different from modern microprocessors (with 8 instruction fetches/cycle, average IPC is only around 1.5). Hence the impact of the fetch logic on the performance was severely underestimated.

Previous way prediction techniques have a common weak point such that the prediction accuracy is heavily dependent on the access history and the penalty of way misprediction is not marginal. Moreover, when the way is mispredicted, all the ways should be reaccessed, resulting in significant energy consumption. On the other hand, our method performs the way determination by accessing the tag array prior to the data array, leading to the best accuracy compared to the previous methods. Our main idea(inserting additional pipeline stage) is similar to [5] that one of us (authors) proposed, but our goal in this paper is to reduce the way misprediction whereas the goal in [5] is to reduce leakage energy.

Table 1 Normalized per-access energy consumption and normalized access latency according to each cache model

	1-way	2-way	4-way	8-way
Per-access energy	1	1.340	1.977	3.205
Access latency	1	1.162	1.166	1.225

3. Proposed Way Determination Technique

3.1 Proposed Pipeline Architecture

We propose a simple and accurate way determination scheme to overcome the shortcomings of previous way prediction techniques by breaking the dependency on the access history. Our key idea is to insert an extra stage, which is called ***tag lookup stage***, between branch prediction stage and fetch stage. During the tag lookup stage, the way is determined by early tag matching. Fig.1(a) shows the conventional pipeline architecture, and Fig.1(b) shows the proposed pipeline architecture with the tag lookup stage to support our scheme. Only the difference of two architectures is the additional tag lookup stage between branch prediction

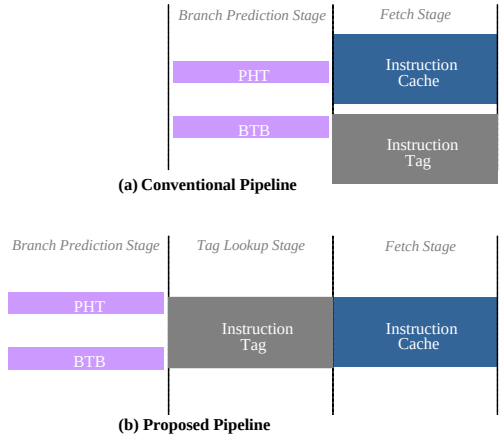

Fig. 1 Comparison of pipeline architectures

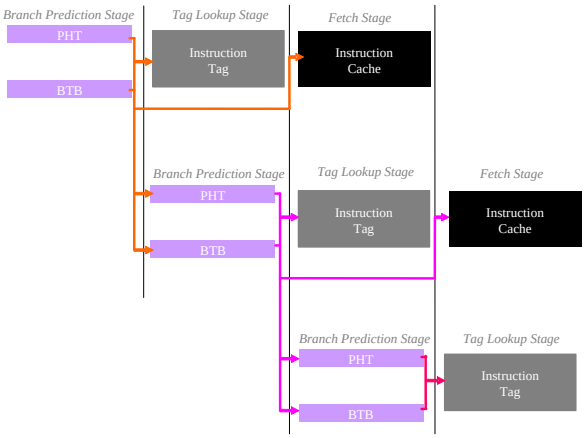

Fig. 2 Pipeline of the proposed way determination when there is no branch misprediction

stage and fetch stage.

As shown in Fig.1(b), in our architecture, where the tag lookup stage is inserted, the branch predictor is accessed one cycle earlier than in the conventional pipeline architecture. ***Note that there is no accuracy decrease in the branch prediction, since the structure of the branch predictor is not changed.*** By inserting the tag lookup stage, we can compare the predicted instruction address with a set of tags during the tag lookup stage, then the way determined is always correct.

Fig.2 depicts the pipeline of the proposed architecture when there is no branch misprediction. The branch prediction stage is moved to one cycle earlier, where the tag matching can be done before the access to the instruction cache (data arrays). Therefore, only one predicted way (data array) is accessed during the fetch stage. Note that there is no way misprediction in the proposed architecture. When the tag is matched during the tag lookup stage, the predicted way is always correct. When the tag is not matched during the tag lookup stage, the instruction cache does not include the instruction (instruction cache miss). In the previous way prediction technique, a cache line is accessed even in the case of a cache miss. On the contrary, in the proposed way predictor, the instruction cache (data array) is not accessed on a cache miss, since the cache miss can be determined prior to the fetch stage by using early tag matching.

3.2 Misprediction Recovery: Difference from the Serial Cache

Generally speaking, additional pipeline stage incurs another branch misprediction penalty. Thus, the serial cache should suffer additional branch misprediction penalty. In the proposed technique, however, this extra penalty is almost hidden. Fig. 3 shows the case that the branch misprediction occurs, where the *instruction* n is the mispredicted instruction. In the conventional pipeline architecture, the instruction cache is accessed for *instruction* $n+4$, right after the writeback stage of the mispredicted branch instruction (*instruction* n). The reason is that an effective address is generated at the writeback stage since it cannot be done in the timing-critical execution stage. This is true in most processors such as Intel-family processors and ARM-family processors. In the proposed scheme, in case of branch misprediction (excluding target misprediction), the tag matching can be hidden, since it is overlapped with the address generation, as shown in Fig.3(b). The key observation is that it is possible to determine the alternative path in parallel with branch resolution. ***For predicted-taken branches, the not-taken path must be woken up (not-taken address is usually carried with the ongoing instruction). For predicted not-taken branches, the taken target is needed. This can either be carried with the instruction or resides in some dedicated storage.*** This capability must exist anyway in current microprocessors because every taken branch in flight must be able to check whether the target address obtained from the BTB is correct or not. Note that the taken target is available at the branch prediction stage regardless of predicted direction, because the direction predictor and target predictor are usually consulted in parallel. ***Since the additional penalty can be hidden in Fig.3(b), there is no difference between the branch prediction penalty of the conventional architecture and that of the proposed architecture. In other words, when the branch misprediction occurs, our architecture does not suffer from an additional penalty. This feature differentiates our technique from the serial cache*** [14] ***where the branch misprediction penalty is always increased by 1-cycle.*** There is only one case when the additional penalty is suffered - branch "target" misprediction. Additional

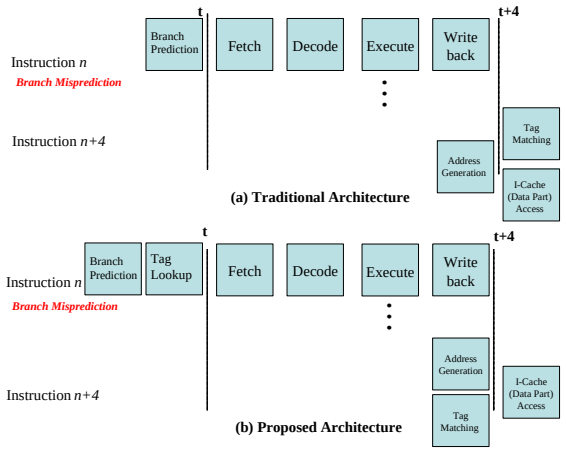

Fig. 3 Comparison of branch misprediction recovery, except the case of branch target misprediction

penalty cannot be avoided in this case. However, the system performance degradation due to branch target misprediction is expected to be little in the proposed architecture, because the branch target misprediction occurs with very low probability.

4. Experiments

4.1 Experimental Environments

In order to evaluate the proposed way determination technique, we compared our technique to the conventional technique where there is no way prediction. Also, we compared it to the previous way prediction technique, similar to [13], which is known as one of the most accurate way prediction techniques. The serial cache that has a two-cycle fetch stage is naturally inferior to any of way predictors that have a one-cycle fetch stage in the perspective of performance. In terms of energy consumption, the serial cache is almost same as the proposed technique. Thus, we do not include the results of the serial cache in this paper.

Table 2 shows the notations for energy models. The number of accesses to the instruction cache consists of the number of correct way determination (or prediction) and twice the number of way misprediction, shown in equation (1). In case of way misprediction, the instruction cache should be re-accessed to fetch the correct instruction. Note that there is no way misprediction in our architecture. The energy consumption of the proposed technique is shown in equation (2). In our technique, the instruction cache does not have to be accessed on an instruction cache miss, since all the tags in a set are looked up earlier and thus cache hit/miss is determined before the instruction cache is accessed. On the other hand, the instruction cache should be accessed in the previous technique regardless of cache hit/miss, since the tag cannot be matched prior to the fetch stage. Even worse, the previous technique should

Table 2 Notations for energy models

Notation	Meaning
N_{I_cache}	Number of instruction cache accesses
$N_{correct\_prediction}$	Number of correct way predictions
$N_{misprediction}$	Number of way mispredictions
N_{cache_miss}	Number of instruction cache misses
$E_{proposed}$	Total energy of the proposed technique
$E_{previous}$	Total energy of the previous technique
E_{1_way}	Energy consumed in 1 way access in the instruction cache
$E_{(N-1)\_way}$	Energy consumed in (N-1) ways access in the instruction cache

re-access the instruction cache in case of way misprediction, which consumes energy to access 1 way and (N-1) ways ('N' denotes cache associativity), shown in equation (3).

$$N_{I\_cache} = N_{correct\_prediction} + 2 * N_{misprediction} \quad (1)$$

$$E_{proposed} = E_{1\_way} * (N_{I\_cache} - N_{cache\_miss}) \quad (2)$$

$$E_{previous} = E_{1\_way} * N_{correct\_prediction} + (E_{1\_way} + E_{(N-1)\_way}) * N_{misprediction} \quad (3)$$

We extended Simplescalar 3.0 [2] to evaluate energy and performance. The processor parameters and energy parameters used in simulations are shown in Table 3. The energy parameters are obtained from Cacti 4.0 Beta [16], based on the 70nm/0.9V technology. We simulated all the applications in SPEC CPU2000 suite [17].

4.2 Impacts on Performance

Fig.4 depicts the ratio which represents the probability that the way is correctly determined without any penalty. Accordingly, the case when the way determination is delayed due to branch target misprediction, is

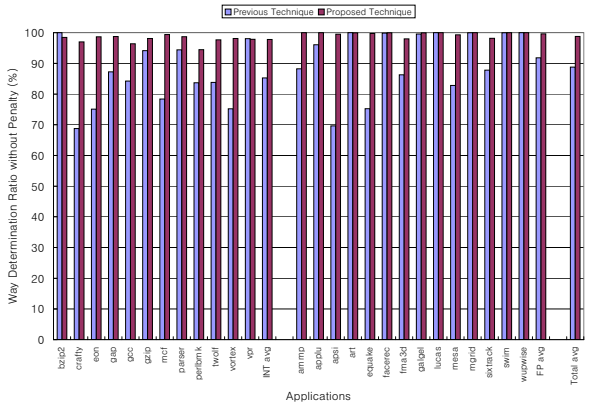

Fig. 4 Ratio of way determination without penalty

Table 3 Simulation parameters

Processor Parameter	**Value**
Instruction Window	64 RUU, 32 LSQ
Fetch/Decode/Issue/Commit Width	4 instructions/cycle
Branch Predictor	Gshare/4K, 1024-entry 4-way BTB
INT FU	4 ALUs, 1 Multi-div
FP FU	4 ALUs, 1 Multi-div
Instruction/Data TLB	128/32 entries in each way, 8KB page size, fully associative, LRU, 1-cycle latency, 28 cycle miss penalty
L1 I-Cache	16KB, 4way, 32B blocks, 1 cycle latency, 4KB sub bank size
L1 D-Cache	16KB, 4way, 32B blocks, 1 cycle latency
L2 Unified Cache	512KB, 4way, 64B blocks, LRU, 12 cycle latency
Energy Parameter	**Value**
Dynamic Energy/1-Way Tag Access	14.23 pJ
Dynamic Energy/4-Way Tag Access	28.10 pJ
Dynamic Energy/1-Way Data Access	14.51 pJ
Dynamic Energy/4-way Data Access	35.55 pJ

the reason for the penalty in the proposed technique. For this reason, the ratio of our technique is lower than 100%. As shown in Fig. 4, the average ratio of the previous technique and that of the proposed technique are 88.8% and 98.8%, respectively. The proposed technique consistently shows higher ratio in all the applications, since the proposed technique determines the way by using early tag matching, which does not depend on the accuracy of a dedicated way predictor. The previous way prediction, which utilizes the access history to predict the way, is highly dependent on the application features, resulting in inconsistent correct way prediction ratio.

As the ratio becomes higher, the performance loss becomes less. Fig.5 represents the execution time normalized to the conventional cache that allows all the ways to be accessed (no way prediction). In some applications, the performance degradation of the previous technique is more than 4%. On average, the previous way prediction technique increases the execution time by about 1%, whereas the proposed technique does only negligible increase (0.07%). Note that even 1% performance degradation in the processor may incur significant ***system-wide*** energy overhead.

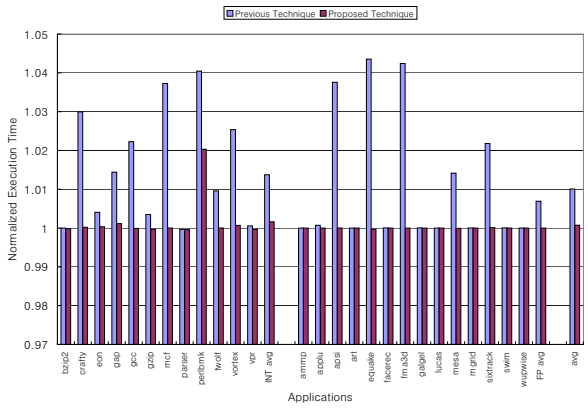

Fig. 5 Execution time, normalized to the conventional instruction cache where there is no way prediction

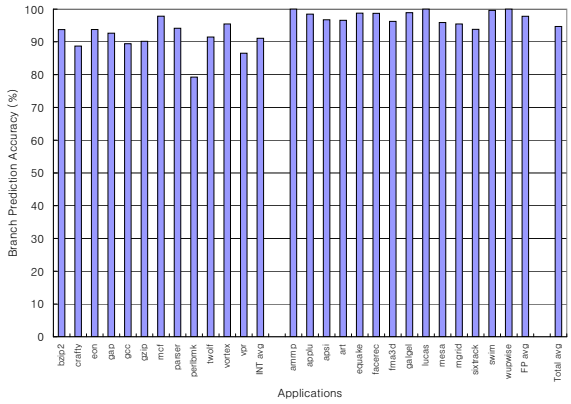

Fig. 6 Branch prediction accuracy

The branch prediction accuracy should be high enough to support the efficiency of the proposed technique. Fig.6 shows the branch prediction accuracy for each application. For example, in perlbmk, the performance loss of the proposed technique is 2.0% (Fig.5), since the branch prediction accuracy is as low as 79.3% (Fig.6). In the other applications, however, the branch prediction accuracy is at least more than 86.5%, resulting in negligible performance loss in the proposed technique.

4.3 Energy Consumption

Fig.7 depicts the energy consumption in the instruction cache (tag and data arrays). The proposed technique reduces the energy consumption in the instruction cache by 55.1% on average compared to that with no way prediction technique, whereas the previous technique reduces 44.6%. In the previous way prediction technique, only one way of the instruction cache (tag and data arrays) is accessed at first, but the tag and data arrays of the instruction cache should be re-accessed simultaneously when the way misprediction

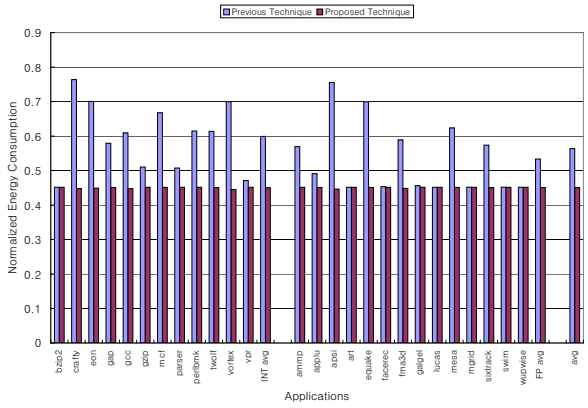

Fig. 7 Energy consumption, normalized to the conventional cache where all the ways in a set are accessed

occurs. On the other hand, in the proposed technique, all the ways of the tag arrays are accessed during the tag lookup stage, and only a single predicted way of the data arrays is accessed during the fetch stage. Hence, there is no case that all the ways of the data arrays are accessed in our architecture, which is a near-optimal solution. Even in case of branch target misprediction, the energy consumption is hardly affected, though there is one-cycle penalty.

Moreover, our technique reduces more energy consumption in case that a cache miss happens, because the instruction cache miss is determined (tag lookup stage) before the instruction cache (data) is accessed (fetch stage).

5. Conclusions

We proposed an accurate and energy-efficient way determination technique for instruction caches, leading to the significant energy reduction with the negligible performance degradation. Our technique inserts an extra pipeline stage, called tag lookup stage, between branch prediction stage and fetch stage. In the tag lookup stage, tag arrays can be accessed early enough to determine the way of the data arrays to access, maintaining performance. The branch prediction accuracy is not deteriorated, since the structure of the branch predictor is not modified in our architecture.

The proposed technique shows negligible performance loss in most applications, whereas the previous way prediction technique severely depends on the accuracy of the dedicated way predictor that is tightly coupled with application features. From the energy perspective, our technique saves more energy in the instruction cache (data and tag arrays) compared to the previous technique. When the proposed technique is combined with more accurate branch predictor such as [6] is adoped, we can have less than 0.07% performance degradation and less energy consumption.

References

[1] D.H. Albonesi, "Selective cache ways: On-demand cache resource allocation," Proceedings of International Symposium on Microarchitecture, pp.70–75, 1999.

[2] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," IEEE Computer Magazine, vol.35, pp.59–67, 2002.

[3] N. Bellas, I. Hajj, and C. Polychronopoulos, "Using dynamic cache management techniques to reduce energy in a high-performance processor," Proceedings of International Symposium on Low Power Electronics and Design, pp.90–95, 2000.

[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architecture-level power analysis and optimizations," Proceedings of International Symposium on Computer Architecture, pp.83–94, 2000.

[5] S. W. Chung and K. Skadron, "On-Demand Solution to Minimize I-Cache Leakage Energy with Maintaining Performance," IEEE Transactions on Computers, Accepted, Available at http://doi.ieeecomputersociety.org/10.1109/TC.2007.70770 .

[6] V. Desmet, H. Vandierendoncka, and K. D. Bosschere, "Clustered indexing for branch predictors ," Microprocessors and Microsystems, vol.31, no.3, pp.168–177, 2007.

[7] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," Proceedings of International Symposium on Low Power Electronics and Design, pp.273–275, 1999.

[8] K. Inoue, T. Ishihara, and K. Murakami, "A high-performance and low-power cache architecture with speculative way-selection," IEICE Transactions on Electron, vol.E83-C, no.2, pp.186–194, 2000.

[9] R. Kessler, "The alpha 21264 microprocessor," IEEE Micro Magazine, pp.24–36, 1999.

[10] S.T. Kim, N. Vijaykrishnan, M. Kandemir, A. Sivasubramaniam, and M.J. Irwin, "Partitioned instruction cache architecture for energy efficiency," ACM Transactions on Embedded Computing Systems, vol.2, no.2, pp.163–185, 2003.

[11] C.H. Kim, S.W. Chung, and C.S. Jhon, "PP-cache: a partitioned power-aware instruction cache architecture," Microprocessors and Microsystems, vol.30, no.5, pp.268–279, 2006.

[12] J. Montanaro and et al., "A 160 mhz, 32b, 0.5w cmos risc microprocessor," Digital Technique Journal, vol.9, no.1, pp.49–62, 1997.

[13] M. Powell, A. Agarwal, T.N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," Proceedings of International Symposium on Microarchitecture, pp.54–65, 2001.

[14] G. Reinman and B. Calder, "Using a serial cache for energy efficient instruction fetching," Journal of Systems Architecture, vol.50, no.11, pp.675–685, 2004.

[15] ARM, "Arm 1136 technical reference manual," http://www.arm.com.

[16] HP, "Cacti 4.0 beta," http://quid.hpl.hp.com:9081/cacti.s

[17] SPEC CPU2000 Benchmarks. http://www.specbench.org.