Power Reduction of a Set-Associative Instruction Cache Using a Dynamic Early Tag Lookup

Chun-Chang Yu
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei, Taiwan
d00943036@ntu.edu.tw

Yu Hen Hu
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
Madison, USA

Yi-Chang Lu
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei, Taiwan

Charlie Chung-Ping Chen
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei, Taiwan

***Abstract*—An energy-efficient instruction cache lookup technique with low area overheads is proposed. The key concept of this Dynamic Early Tag Lookup (DETL) method is to exploit the presence of instruction fetch-bubble cycles. In a fetch-bubble cycle, the index of the matching cache set can be determined earlier. Hence, the dynamic energy for parallel memory accesses to irrelevant cache banks can be saved. We implemented the proposed DETL algorithm on a 4-way set-associative instruction cache in a RISC-V micro-architecture, and tested its performance using the SPEC CPU2006 benchmark suite. The experiment results showed a 19.38% dynamic power reduction with an area overhead smaller than 0.1%.**

***Index Terms*—dynamic early tag lookup, energy-efficient, instruction cache, processor architecture**

I. Introduction

Dynamic power reduction is a key design objective of low power micro-architecture for artificial intelligence of things which are cost and power sensitive. One of the best candidates to reduce dynamic power is the instruction cache (I-cache), which is activated constantly whenever the processor is turned on. The I-cache is often organized to support multi-way set-associative cache access to balance cache misses and costly associative tagging. To reduce the latency of a cache lookup, the traditional design of the I-cache requires simultaneous access to all ways of memory banks while performing the tag decoding. The decoded tag index then selects only one of the memory bank outputs as the fetch result, wasting dynamic power for accessing the remaining memory banks. A different I-cache design may perform tag decoding first, and use the decoded tag to access the matched memory bank only. However, this sequential approach would increase the I-cache latency by one cycle for all instructions.

Way prediction methods [1]– [3] were proposed to predict the matching way (tag index), and access only the matched memory bank. The penalty of misprediction, however, may overshadow the potential power saving, and the hardware (chip area) overhead can be overwhelming. In HotSpot [4], a fully associative L0 cache is implemented to cache frequently accessed instructions. However, cache miss penalties still exist and the L0 hardware overhead is huge. In a tagless cache approach [5], way index information is incorporated in the translation lookaside buffer (TLB). But this approach increases cache access latency, since the TLB needs to be accessed before the I-cache lookup, and the size of the TLB is significantly increased in this scheme. In the early tag lookup scheme [6], an early tag lookup will be performed when calculating the next program counter (PC) address. Based on the early lookup result, only the matched memory bank will be accessed. However, the existing early tag lookup scheme requires doubling the read ports of both the I-cache tag and the branch target buffer (BTB), so that the tag and the buffer can be accessed simultaneously.

In this paper, we present a novel approach that achieves significant dynamic power reduction in the I-cache with negligible hardware overheads for low resource budget embedded processors. We name our method the Dynamic Early Tag Lookup (DETL). In the traditional I-cache architecture, the instruction branch detection unit is integrated with the I-cache. When the instruction queue is full, the instruction-fetch unit stalls, leaving a fetch-bubble clock cycle. With the DETL, we decouple the instruction branch prediction unit from the I-cache by inserting a lookahead program counter queue between them. Thus, during a fetch-bubble cycle, the branch prediction unit continues to predict the next PC value, and the I-cache performs an early tag lookup to determine the matched cache memory bank index (way) or detect a cache miss. When the instruction fetch unit resumes in the following cycle, the I-cache proceeds to access the matched cache memory bank or handle the cache miss without simultaneously accessing the remaining cache memory banks. As such, a significant amount of dynamic power consumption for accessing those cache memory banks is saved. The only hardware overhead in this scheme is the inserted lookahead program counter queue, which is negligible compared to other methods [1]– [6]. We implemented the DETL I-cache in a RISC-V micro-architecture [7] using the GEM5 simulator [8] with 28nm technology parameters. Simulation results indicate a 19.38% dynamic power reduction with the area overhead smaller than 0.1%.

II. Dynamic early tag lookup

A. Possibility for the early tag lookup

Performance and energy are two major trade-off factors in computer architecture. In the I-cache, there are two essential schemes to get high performance or low energy. First, parallel

978-3-9819263-5-4/DATE21/©2021 EDAA

access to the I-cache tag RAM and data RAM can achieve high performance. Its access time is the sum of the maximum delay of tag/data RAM accesses and multiplexer delay to select data from the set-associative data RAM. However, parallel access consumes a large amount of energy to read out multiple data for selection. The energy, *E(P)*, and latency, *L(P)*, of parallel access are listed as follows.

$$\begin{aligned} E(P) &= N \times E(Tag) + N \times E(Data) \\ L(P) &= max(L(Tag), L(Data)) + L(Multiplexer) \end{aligned} \quad (1)$$

where *N* and *max* mean the number of ways and the maximum value of the options, respectively. The second method is to access the tag RAM and data RAM in sequence. It consumes low energy since the target way of data RAM is determined by the tag RAM lookup. Sequential access takes an accumulated delay of the tag RAM and data RAM. The energy, *E(S)*, and latency, *L(S)*, of sequential access are listed as follows.

$$\begin{aligned} E(S) &= N \times E(Tag) + E(Data) \\ L(S) &= L(Tag) + L(Data) + L(Multiplexer) \end{aligned} \quad (2)$$

The best solution is to maintain high performance and consume low energy. This goal may be achieved if we can early look up the tag RAM to determine the way in the fetch-bubble cycles. In the following cycles, the tag RAM and data RAM are accessed in parallel, but for different fetch blocks. The energy, *E(ETL)*, and latency, *L(ETL)*, of the ETL mode are listed as follows.

$$\begin{aligned} E(ETL) &= N \times E(Tag) + E(Data) \\ L(ETL) &= max(L(Tag), L(Data)) + L(Multiplexer) \end{aligned} \quad (3)$$

Before implementation, we estimate the feasibility of the early tag lookup in benchmark simulations. The simulation of a pipeline is being performed. The pipeline includes a limited-entry instruction queue (IQ) in between fetch stages and decode stages. IQ is a pointer-based circular buffer, which is decoupled from the rest of stages. The stall from the back end pipes does not directly affect the instruction-fetch unit (IFU). When the IQ is full, there are fetch-bubble cycles. During the fetch-bubble cycles, the I-cache unit is idle. Since the I-cache is idle for the current fetch block, the I-cache tag can be looked up for the next fetch block before the IQ is ready to be filled. If the lookup results in a hit, the exact matching way of the I-cache has been determined before exiting the bubble cycles. When the IFU resumes activation, the I-cache can only turn on the matching way of the data RAM. If the lookup results in a miss, a miss handling request is issued, while the data RAM is not necessary to be turned on. The simulations in benchmarks show that the IFU has a 12.8% to 17.4% possibility to get a fetch-bubble cycle for the early tag lookup. Once a tag of a fetch block is accessed earlier in the bubble cycle, the shifted parallel access can be applied for the following fetch blocks until exceptions or redirect events. The early tag lookup is terminated to minimize the resume/misprediction penalty. Based on this scheme, it is possible to reduce energy without sacrificing performance.

B. Implementation of the DETL in the I-cache

The early tag lookup is dynamically enabled once the IFU is idle. The idle fetch state results in fetch bubbles in the pipeline.

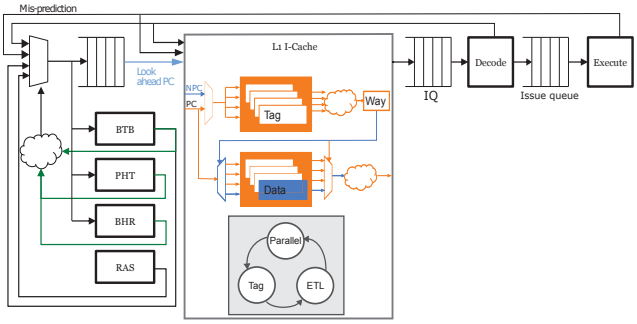

Fig. 1. DETL Micro-architecture

The IQ full condition and I-cache miss lead the idle state for one or multiple cycles. In this paper, we focus on the IQ full condition. From reset, exception or redirect events, the DETL monitors the first idle fetch state and then switches to the early tag lookup (ETL) mode for the following cycles until exceptions or redirect events occur. For multiple stall cycles, the energy saving is the same as the one-cycle stall. During the IQ stall cycles, the I-cache data RAM is inactive in both the parallel scheme and our proposed idea. We save energy of the data RAM in the effective fetch cycles following the first IQ stall cycle.

The micro-architecture of the DETL as illustrated in Fig. 1 separates the IFU into the branch prediction unit and the L1 I-cache unit. The branch prediction unit provides the direction predictions and target predictions of control transfer instructions in program sequences. The prediction results, including the PC and the next PC (NPC), are stored in a queue in front of the I-cache unit. Based on the predicted PC, the I-cache unit looks up the instruction data and handles cache miss operations. When the fetch results are returned, the instruction data are stored in the IQ behind the I-cache unit. The IQ in between the I-cache and decoder is necessary to provide sufficient instructions to be passed to the back end pipes in superscalars. The IQ is full as long as the back end pipes are stalled by long latency instructions, D-cache misses, or hazards. While the IQ is full, the IFU should not be feeding the instruction data into it. These cycles are fetch bubbles that we are going to use for early tag lookup. In the ETL mode, the NPC and PC are selected to look up the I-cache tag and data, respectively.

At the point of the tag RAM, a two-to-one multiplexer is required to multiplex the address bus. The I-cache lookup mode provides control for the multiplexer. At the point of the data RAM, a distributor and a multiplexer are required. The distributor provides a *Select* signal for each way of the I-cache. The *Select* signal is decoded from the matching way and the operation mode of I-cache. The decode logic also provides the multiplexer with the *Select* signal to enable the multiplexer to route the appropriate instruction data, from the selected data RAM to the IQ. As the blue lines shown in Fig. 1, only the matching way of the data RAM is accessed using the address from the PC, since the way information has been stored in a register in the previous cycle. In the parallel mode, both tag and data choose the PC as the lookup index. The hardware

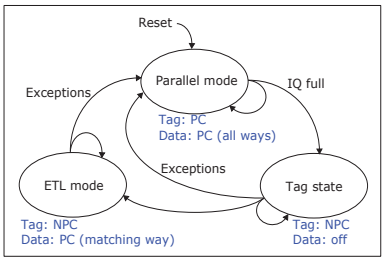

Fig. 2. DETL Control Path

module labeled as *Way* preserves the matched tag information. The distributor in front of the data RAM is deployed in the ETL mode. The multiplexer behind the data RAM is deployed in the parallel mode. Data paths of the parallel mode and the ETL mode are dynamically switched by the control path.

The control path of the DETL is illustrated in Fig. 2. After reset, the I-cache is in the parallel mode.

- Parallel mode: All ways of the tag and data RAMs are accessed at the same time for the same fetch PC. The multiplexer behind the data RAM chooses the data of the matching way. The fetch results are filled into the IQ for decoding in order. Once the IQ is full, the I-cache dynamically turns to the Tag state.
- Tag state: The NPC is used to look up the tag RAM during the fetch bubbles, while the data RAM is off. If the tag lookup results in a hit, the matching way is determined. Once the bubble is released, the I-cache turns to the ETL mode.
- ETL mode: The PC is used to look for the data RAM of the matching way. At the same time, the NPC is used to look up the tag RAM for the next PC. In the following cycles, the tag is looked up for the NPC, while the data RAM is served for the PC in parallel. The data RAM is only turned on the matching way instead of all ways. The processor stays in the state of ETL mode until exceptions or redirect events occur. The reason we switch the I-cache back to the parallel mode in redirect events is to maintain the high performance.

C. Look-ahead prediction PC

The DETL must cooperate with the look-ahead PC, since it looks up the tag of the next fetch block when the pipeline is stalled by the IQ. This look-ahead PC is generated by branch predictors (BP). Based on this requirement, a queue between the BP and the IFU is necessary to provide consecutive look-ahead PCs. This queue decouples the BP from the IFU [9]. The BP can go proceed without being blocked by the pipeline. The BP including the BTB, prediction history table, branch history register, and return address stack provide the correct look-ahead PC for the I-cache, since the branch misprediction would terminate the DETL scheme. The data path from misprediction to the I-cache improves the penalty cycles. When misprediction occurs, the I-cache can be accessed before the BP gets results. The redirect target PC is used to look up I-cache for the instruction data at the same cycles as the misprediction happens. Therefore, there is no performance overhead from introducing the look-ahead PC queue.

TABLE I
PROCESSOR CONFIGURATION

Fetch width	8 bytes	Pipeline stages	8
Issue/Commit width	2	Function units	1 ALU, 2 MUL, 1 FPU
Instruction/Decoded Stream buffer size	64/16	L1 I/D-cache	64KB, 4-way, 32B cache line
Branch predictor	Gshare 8K PHT	BTB/RAS size	4K/16

III. EXPERIMENT

A. Methodology

The DETL has been implemented on a Gem5 [8] simulator. The simulator configuration is a RISC-V ISA and an in-order, dual-issue architecture. The RISC-V ISA is a base-line instruction set plus compressed extension. The fetch block is 64 bits wide. A fetch block includes down to two 32-bit instructions and up to four 16-bit instructions. As a result, the IQ between the fetch stage and the decode stage can accumulate sufficient instructions for the execution units. The processor configuration is listed in Table I. Gem5 simulated with the SPEC CPU2006 [10] benchmark, which is a CPU-intensive benchmark suite for stressing a processor. Workloads developed from real user applications provide a comparative measure of compute-intensive performance. Maintaining the performance metric of the instruction per cycle is our research purpose.

MCPAT [11] is deployed to estimate dynamic power consumption in 2 GHz with 28nm technology. The power comparison is based on the same frequency, capacity, and voltage. Different cache access schemes have different cache activities, which are the metrics for power consumption. The dynamic power estimation utilizes the activities evaluated by simulating SPEC CPU2006 benchmarks on the Gem5 simulator.

B. Results

The baseline design is an in-order dual-issue 32-bit processor in parallel I-cache access micro-architecture. The average amount of the I-cache data RAM accesses in the DETL becomes 41.54% of the baseline design. The distribution of the ratios in seven workloads is illustrated as the green bars in Fig. 3. The best case is 33.54% in the h264ref workload about video stream encoding, and the worst case is 46.8% in the libquantum workload about polynomial-time factorization of quantum computing. The idle cycle rate in h264ref is 22.18% which is higher than 5.3% in libquantum, and the higher idle cycle rate in h264ref gives more opportunities to switch to the ETL mode after a branch misprediction. In the cases with higher I-cache miss rates, such as h264ref, the DETL can avoid unnecessary data RAM accesses since I-cache misses can be detected earlier. The results show that the DETL efficiently reduces the amount of data RAM accesses. On the other hand, the DETL reduces all-way data accesses to the I-cache by 79% on average. The reduction of data access can be observed as indicated by the green bars in Fig. 4 for different benchmark workloads. The DETL gets the most reduction in all-way data accesses by 88% in the h264ref workload. The smallest reduction is 71% in the libquantum workload.

Dynamic power reduction of the I-cache is shown as the blue bars in Fig. 4. The DETL reduces 19.38% dynamic power on

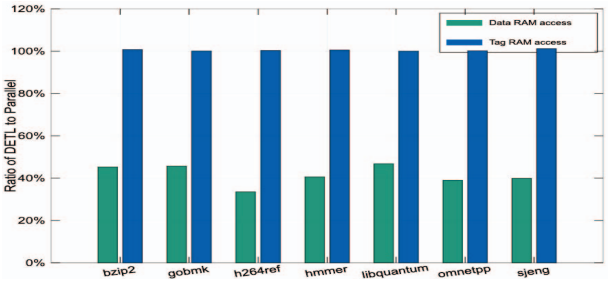

Fig. 3. The access ratio of Data/Tag RAM in DETL to parallel I-cache

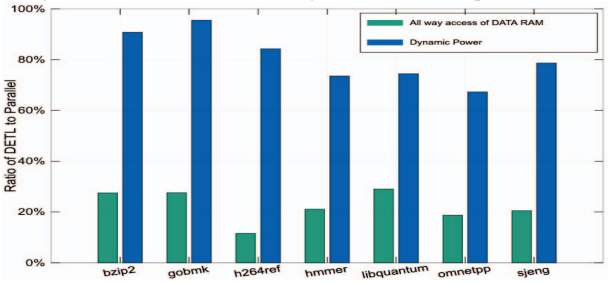

Fig. 4. The ratio of dynamic power in DETL to conventional I-cache in parallel

average. The best case is a 32.76% reduction in the omnetpp workload about discrete event simulation of a large Ethernet network. The worst case is a 4.52% reduction in the gobmk workload about the game playing of artificial intelligence.

C. Overhead

The DETL does not have performance overhead, since there is no penalty in the speculatively earlier tag lookup. The ETL mode begins in the fetch-bubble cycle and is maintained in the following fetch cycles until a redirect event. Instruction data returns from the I-cache data RAM in the same cycle as the baseline design. Compared to the total energy saved, shown in Fig. 4, the energy overhead of the DETL controller itself is very small (0.42% on average). The overhead comes from speculative tag accesses for the bad paths that will be redirected once the branches are evaluated. The area overhead of the DETL is from the look-ahead PC queue, additional logic of multiplexers in the data path, and extra states to switch the parallel mode and ETL mode in the control path. This area overhead is negligible in the processor.

D. Comparison

In addition to the comparison with the baseline design, the DETL is competitive with the previous works as shown in Table II. The data sources are also shown in the first column (see the references for implementation and experimental details). The DETL is the most lightweight in area overhead while all-way access reduction achieves 79% without trading off performance. Previous ETL schemes have area overhead not only in multiple ports of the tag RAM but also in the BTB and prediction history table. HotSpot introduces a two-level I-cache architecture. The tagless cache and way prediction need a huge TLB and a large way prediction table, respectively. Both the DETL and ETL can achieve energy reduction without sacrificing performance. ETL achieves higher I-cache access reduction in the cost of larger area overhead.

TABLE II
OVERHEAD COMPARISON WITH PREVIOUS WORKS

Scheme Name	Performance overhead	Area overhead
DETL	None	Negligible
ETL [6]	None	Additional read ports in I-cache tag, BTB, and BP
TLC [5]	Mild (I-cache access latency is increased)	Significantly increases TLB size
HotSpot [4]	Mild (Additional I-cache access penalty)	Additional L0 I-cache structure
Way Prediction [1]– [3]	Medium (Additional I-cache access penalty)	Heavy way predictor tables

IV. CONCLUSION

In order to improve the dynamic power consumption of the set-associative I-cache without performance loss and area overhead, we propose the DETL scheme to look up the data RAM in a matching way instead of all ways. The I-cache is dynamically switched from the parallel mode to the early tag lookup mode when there is a fetch bubble. Before pipeline redirect events occur, only the matching way needs to be accessed since it is determined one cycle earlier. We have observed that fetch stages are stalled frequently in modern processors that fetch multiple instructions in a cycle. Gem5 simulation shows that the DETL reduces the amount of the I-cache data RAM accesses to be 41.54% of the amount in the parallel scheme. As a result, dynamic power is reduced by 19.38% in 28nm technology. This energy improvement is realized with the area overhead less than 0.1%.

REFERENCES

[1] W. Tang, A. Veidenbaum, A. Nicolau, and R. Gupta, "Integrated i-cache way predictor and branch target buffer to reduce energy consumption," in Int'l Symp. on High Performance Computing, pp. 120–132, Springer, 2002.

[2] M. D. Powell, A. Agarwal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," in Proc. of the 34th Annual ACM/IEEE Int'l Symp. on Microarchitecture, pp. 54–65, 2001.

[3] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in Proc. of the Int'l Symp. on Low Power Electronics and Design, pp. 273–275, IEEE, 1999.

[4] C.-L. Yang and C.-H. Lee, "Hotspot cache: joint temporal and spatial locality exploitation for i-cache energy reduction," in Proc. of the Int'l Symp. on Low Power Electronics and Design, pp. 114–119, ACM, 2004.

[5] A. Sembrant, E. Hagersten, and D. Black-Shaffer, "Tlc: A tag-less cache for reducing dynamic first level cache energy," in Proc. of the 46th Annual IEEE/ACM Int'l Symp. on Microarchitecture, pp. 49–61, IEEE, 2013.

[6] W. Zhang, H. Zhang, and J. Lach, "Reducing dynamic energy of set-associative l1 instruction cache by early tag lookup," in IEEE/ACM Int'l Symp. on Low Power Electronics and Design, pp. 49–54, IEEE, 2015.

[7] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for risc-v," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146, 2014.

[8] N. Binkert, B. Beckmann, and et al., "The gem5 simulator," SIGARCH Comput. Archit. News, vol. 39, pp. 1–7, Aug. 2011.

[9] G. Reinman, B. Calder, and T. Austin, "Optimizations enabled by a decoupled front-end architecture," IEEE Trans. on Computers, vol. 50, no. 4, pp. 338–355, 2001.

[10] J. L. Henning, "Spec cpu2006 benchmark descriptions," SIGARCH Comput. Archit. News, vol. 34, p. 1–17, Sept. 2006.

[11] S. Li, J. H. Ahn, R. D. Strong, and et al., "Mcpat: An integrated power,area, and timing modeling framework for multicore and manycore architectures," in Proc. of 42nd Annual IEEE/ACM Int'l Symp. on Microarchitecture, pp. 469–480, Dec. 2009.