WHOLE: A Low Energy I-Cache with Separate Way History

Zichao Xie, Dong Tong, Xu Cheng
Microprocessor Research & Development Center, Peking University, Beijing, China
{xiezichao, tongdong, chengxu}@mprc.pku.edu.cn

Abstract— **Set-associative instruction caches achieve low miss rates at the expense of significant energy dissipation. Previous energy-efficient approaches usually suffer from performance degradation and redundant extension bits.**

In this paper, we propose a Way History Oriented Low Energy Instruction Cache (WHOLE-Cache) design for single issue and in-order execution processors. The WHOLE-Cache design not only achieves a significant portion of energy reduction by effectively reducing dynamic energy dissipation of set-associative instruction cache, but also leads to no additional cycle penalties. Tag comparison results are stored into either the Branch Target Buffer (BTB) or the Instruction Cache (I-Cache) to avoid tag checks and unnecessary way activation for subsequent accesses to visited cache lines. The extended BTB uses way history bits for branch instructions, while the I-Cache extension bits are used in case of fetching consecutive instructions resided in different cache lines. A valid flag is associated with each stored tag comparison result to indicate whether the instruction to be fetched is resided in the recorded location. A simple invalidation scheme is implemented in the cache miss replacement operation. Whenever a cache line is replaced, the pointers to it, which reside in the BTB or other I-cache lines, will be invalidated accordingly.

We model the WHOLE-Cache design in Verilog. By deriving basic parameters from TSMC 65nm technology, we use Wattch simulator to evaluate the performance and energy reduction of the WHOLE-Cache in the instruction fetch stage. We use SPEC2000 and Mediabench as benchmarks. It is observed that compared with a conventional 4-way set-associative I-Cache, the energy consumption of the WHOLE-Cache is reduced by 65% without any performance penalty.

I. INTRODUCTION

On-Chip caches have become one of the most energy-consuming components in a modern microprocessor, especially the instruction cache because of its high access frequency. Modern microprocessors usually employ set-associative caches to reduce conflict miss rate [8], [11]. However, an N-way set-associative cache tends to dissipate almost N times the amount of energy as the access of a direct-mapped cache, as it probes multiple tag and data arrays in parallel, but only the one with a tag match will be used. On the other hand, as we enter the multicore era, the challenge is to increase performance without increasing energy per operation or silicon area. Shallower pipelines with in-order execution processor becomes one of the basic building blocks and has proven to be the most energy- and area-efficient [14]. Therefore, the related low energy design to this kind of processor becomes more important than before.

There are several existing techniques that aim to reduce instruction fetch energy in set-associative caches. Way Prediction [6], [9] uses a way-prediction table to access the tag and data of the predicted way only. Way Memorization [5], History-Based Tag Comparison Cache [7] and Tagless Hit Instruction Cache [10] record the previous tag comparison results to eliminate the way search operation.

However, it is important to notice that those approaches concentrate the low energy design only in one structure, instead of considering exploiting other existing structures together. This usually brings about redundant bits which may reduce the efficiency of the low energy structures. And the operation conflicts on the history bits will also lead to performance degradation. Moreover, complex operations prevent them from being used in industry.

In this paper, we propose a way history oriented low energy instruction cache design which targets single issue and in-order execution processors. The idea is to leverage on an extended BTB and an extended I-Cache to avoid I-Cache tag and way accesses for branch and sequential instructions respectively. In particular, tag-comparison results, which will be re-used for selecting the hit-way, are stored either in additional bits in BTB for branch instructions or in the additional I-Cache related array for sequential instructions resided in different cache lines. A valid flag is associated with each stored tag comparison result to indicate whether the target instruction is resided in the recorded location. In essence, the WHOLE-Cache acts as a direct-map cache without tag lookup for those instructions which have not been evicted from the cache since their last accesses. In a case where an instruction is accessed for the first time, or it has been evicted from I-cache, it will be fetched in the conventional way without any delay.

This paper makes the following contributions. Firstly, the WHOLE-Cache not only achieves significant dynamic energy reduction, but also leads no additional cycle penalties. Conflicts eliminations and latency hiding could be implemented because this energy-efficient design is not just implemented in the I-Cache, but separately in the I-Cache and in the BTB. Secondly, the intuition behind this structure is to add minimum useful storage bits to implement the low energy design in 65 nm technology. Furthermore, the cache structure and the hierarchy of memory system are maintained. And the extended bits can be implemented in individual arrays, so they can be accessed in parallel and does not affect the original cache structure. This is considered an advantage in industry because it makes it possible to use the processor core with previously designed caches.

The rest of this paper is organized as follows: Section 2 shows related work; Section 3 presents the organization

978-1-4244-5028-2/09/$25.00 ©2009 IEEE

of the WHOLE-Cache design and explains its operation in detail; in Section 4, we evaluate the energy efficiency and compare it with the related techniques; we conclude in Section 5.

II. RELATED WORK

There are many existing techniques proposed to alleviate the negative effect of set-associative instruction caches.

A well-known approach is to employ a filter or L0 instruction cache between the microprocessor and the L1 main cache [4], [15]. The filter or L0 caches are small and typically direct-mapped for the purpose of providing more energy-efficient access to frequently fetched instructions. Although the frequency of an L1-Cache is reduced, any miss in L0-Cache incurs additional miss penalties. Though [10] made an improvement, the penalties can be accumulated and result in significant performance degradation for some applications.

Way prediction [6], [9] is a speculative technique for reducing dynamic energy of the instruction cache. By accessing only a single predicted way, instead of all the ways in a set, energy consumption can be reduced. If the way prediction is not correct, an extra cycle is required to perform tag comparisons of all ways. Even if the prediction is correct, this technique must also read the tag array to verify the prediction. Our approach does not require the checking process and only read out the data when the tag comparison results are used.

Compared with the way prediction approach, the method of using the cache tag comparison results to achieve the energy reduction is more suitable for instruction caches, because the instruction flow is organized regularly in the conventional instruction caches. Ma et al. [5] proposed a dynamic way memorization technique which eliminates the way search operations. The idea is to record both the tag check results and the valid bits within I-Cache. Valid flag guarantees that the next sequential line accessed or the line associated with the target of a branch is in the I-Cache. The downside of this approach is that it requires extra fields for each instruction. In our approach, the tag comparison re-use for the branch instruction is recorded in the BTB, which is to save area and static energy brought by unusually-used bits.

The history-based tag-comparison cache (*HBTC Cache*) is a related approach to reduce the number of tag checks and avoid the unnecessary way activation. It extends the BTB to store an execution footprint which indicates whether the next instruction to be fetched resides in the L1 I-Cache [7]. Unlike the HBTC, our design handles both the branch case and the case of consecutive instructions residing in different cache lines. And the BTB is only recorded for the taken branches.

Compared with other related methods, we believe that the most energy-efficient instruction fetch structure is a structure developed by utilizing inherent properties of the existing components. This is because this principle usually requires simpler implementation logic and smaller hardware overhead. The latter feature is quite suitable for the future technology generations. The WHOLE-Cache we study in this paper exploits the BTB inherently employed for branch prediction and the I-Cache inherently employed for consecutive instruction fetch to memorize the tag comparison results of the I-Cache.

III. WAY HISTORY ORIENTED LOW ENERGY INSTRUCTION CACHE (WHOLE-CACHE)

This section presents an overview to the Way History Oriented Low Energy Instruction Cache (*WHOLE-Cache*). Firstly, we describe the principle of tag comparison re-use as well as the intuition of the I-Cache and BTB separate extension structure. Secondly, we explore the invalidation operations required for the WHOLE-Cache. Finally, we analysis its performance impact.

A. The Principles of the WHOLE-Cache Design

Tag comparison re-use is the basic principle of the WHOLE-Cache. Instructions in the I-Cache are usually executed repeatedly. At the first reference of the instruction, the tag check has to be performed (*Fetching Normally*). However, at and after the second reference, if no cache miss has occurred since the first reference, it is guaranteed that the target instruction currently resides in the same location of I-Cache. Therefore, tag comparison result could be used to avoid unnecessary way activation and save tag lookup energy (*Fetching Directly*).

The other key principle in the WHOLE-Cache design is the idea of extending way history bits to I-Cache and BTB respectively. Respective extension is based on the following three possibilities when the instructions are fetched: 1) successive instructions fall within the same cache line, 2) successive instructions reside into different cache lines, and 3) the target instruction is fetched behind a taken branch. We call these three cases as *sequential-intra-line flow*, *sequential-inter-line flow* and *non-sequential flow*, respectively. In the first case, it is possible to identify the way number for the latter instruction by memorizing the way number of the previous instruction. This is because the current word is guaranteed to be found in the same cache line as the previous one. For the sequential-inter-line flow, we store the tag comparison results of the latter instruction into the cache line of the previous instruction by using additional memorization bits related to the I-Cache. This is because the I-Cache is an inherent structure for fetching instructions consecutively. By the same token, BTB has been used as a mechanism for effectively predicting the behavior of a branch [12]. Therefore, the BTB is extended by storing tag comparison result of the target instruction in case of the non-sequential flow. Return Address Stack (*RAS*) provides the return address for function returns. We could also extend the RAS to save tag comparison result. Since the mechanism and operations of the extended RAS are as the same as those of the BTB, BTB we mention in the following sections stands for both the BTB and the RAS.

B. The WHOLE-Cache Structure

Fig. 1 shows the detailed view of a WHOLE-Cache design. The first aspect to notice in the WHOLE-Cache

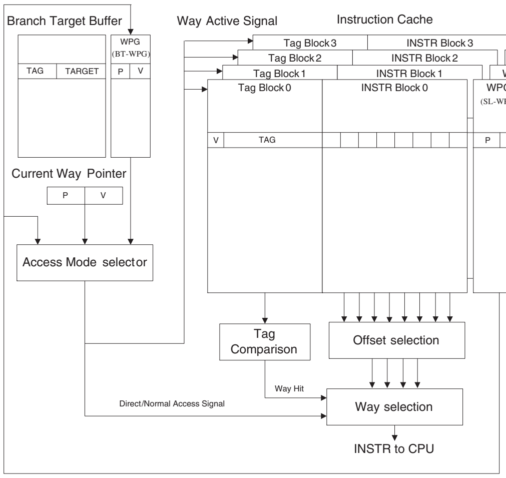


Fig. 1. Way History Oriented Low Energy Instruction Cache Design

is the presence of *Way Pointer Group* (*WPG*). A WPG basically consists of *a way pointer* and *a valid flag*. A way pointer contains the tag comparison result (hit-way number) to specify the hit way of the corresponding instruction, so it can be implemented as a log *n*-bit flag where *n* is the cache associativity. The 1-bit valid flag is used for determining whether the way pointers are valid. Together, a WPG forms a unique way location in the I-Cache which is guaranteed to hold the desired instructions when the valid flag is set. For simplicity, we use *Sequential-Line-WPG* (*SL-WPG*) for way pointer groups used in the I-Cache and *Branch-Target-WPG* (*BT-WPG*) for those used in the BTB. A single WPG is added into the instruction fetch unit which is also called *Current Way Pointer* (*CWP*). The CWP memorizes the way number of the current fetched instruction for the use of sequential-intra-line flow. *AccessModeSelector*, a 3to1 muxiplier, is added to the datapath of instruction fetch, which is for selecting which WPG to be used between the CWP, the SL-WPG or the BT-WPG.

If the BTB specifies a taken branch, the WHOLE-Cache will make use of the BT-WPGs, each of which is associated with each BTB entry. If the fetching branch instruction has its valid flag set, then the branch target's cache line is guaranteed to be available to access the matched way directly. If the valid flag is not set, then the target instruction should be fetched conventionally from the I-Cache instead. After fetching target instruction normally, the BT-WPG should be updated so that the entry of the branch instruction now records the useful way pointer.

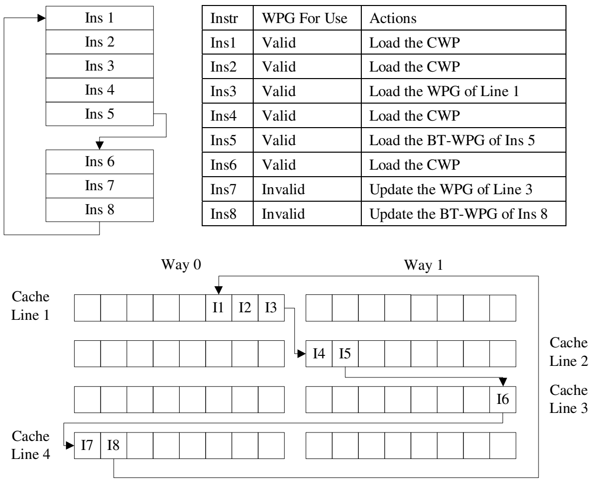
In a case of a sequential fetch access, including a case where the branch prediction is not taken, sequential-intra-line flow and sequential-inter-line flow have to be considered. If we are accessing any instruction other than the last one in the cache line (sequential-intra-line flow), then we will choose to fetch the next instruction directly from I-Cache when the valid flag of the CWP is set. The way pointer in the CWP will still be available, or the CWP will be updated by the tag comparison result of the current instruction. If it is the last instruction in the cache line that is being fetched (sequential-inter-line flow), fetching the next instruction directly will occur only if the SL-WPG of the current cache line is valid. A valid SL-WPG signifies that when the next cache line is accessed, the matched way can be directly accessed without tag lookup. If the SL-WPG is invalid, the tag comparison result of the next cache line should be memorized in the SL-WPG of the current cache line. Normally, if a valid way pointer is found in a non-sequential flow or a sequential-inter-line flow, the content of the related BT-WPG or SL-WPG would be written into the CWP for future use.

Fig. 2 shows an example that illustrates how the WHOLE-Cache works. There are eight instructions spanning two basic blocks. At the beginning, we assume that the CWP contains the valid way number of cache line 1, so when instruction 1 is fetched, the CWP is used for performing direct way access. Instruction 2 and 3 are fetched and are guaranteed to read the matched data word only since they are sequential references within the same line. When fetching instruction 3, it is found that instruction 3 is the last instruction in cache line 1 and is not a branch. This determination could be performed by checking the lower bits of its PC and accessing BTB simultaneously. Cache line 1's SL-WPG is valid at this point, indicating that the next sequential cache line now has the previous tag comparison result to be used. Therefore, instruction 4 is fetched directly, while the way number of cache line 2 is loaded into the CWP. Instruction 5 is a predicted taken branch. By accessing BTB, its related BT-WPG is valid to reflect that the target of instruction 5 (instruction 6) can also be accessed directly. However, the SL-WPG's valid flag of cache line 3 is not set, so we have to fall back to perform conventional cache access to instruction 7. After the tag comparison process is finished, the hit way number is stored into both the SL-WPG of cache line 3 and the CWP. Instruction 8 is another predicted taken branch but with invalid BT-WPG. Therefore, its target instruction, instruction 1, has to be fetched normally. Then the matched way number is saved into both the BT-WPG and the CWP. After the above process, the instructions fetched in the remaining iteration of the loop are totally guaranteed to be accessed directly.

C. Invalidation Scheme

When a cache line is evicted, the CWP is surely invalidated and there may be some SL-WPGs and BT-WPGs which need to be invalidated for they point to the victim cache line. If not, when they are traversed, instructions from the wrong cache line will be fetched. In this section, we present our invalidation policy of the WHOLE-Cache, which takes both the implementation complexity and efficiency into consideration.

We take the invalidation of the SL-WPGs into consideration first. A direct method (*Exact Invalid Scheme*) is using tag search to locate the previous word and then invalidate the SL-WPG of the matched way. This strategy requires tag comparison operations in cache miss process. An alternative strategy, *Conservative Invalid Scheme*, is simpler. All the



Instr	WPG For Use	Actions
Ins1	Valid	Load the CWP
Ins2	Valid	Load the CWP
Ins3	Valid	Load the WPG of Line 1
Ins4	Valid	Load the CWP
Ins5	Valid	Load the BT-WPG of Ins 5
Ins6	Valid	Load the CWP
Ins7	Invalid	Update the WPG of Line 3
Ins8	Invalid	Update the BT-WPG of Ins 8

Fig. 2. An Example of Using the WHOLE-Cache

WPGs of the previous cache set will be invalidated no matter which way is matched. The tag comparison and the invalidation to the valid bits of upper set can both be hidden into the cache miss process, so these two schemes would not lead to performance degradation.

There are at least three invalidation schemes for the BT-WPGs. Fig. 3 shows three sample line configurations for the WHOLE-Cache. Fig. 3 (a) illustrates the simplest method, *General Invalid Scheme*. Whenever any cache line is evicted, all the WPG in BTB should be invalidated. Fig. 3 (b) shows the 0-*Target scheme*. A *target bit* is attached to each cache line to indicate whether a branch is created to this cache line. The target bit is set during the BTB updating. Only if a cache line with a target bit set is replaced, all the BT-WPGs will be invalidated. Otherwise, it is unnecessary to clear all the BT-WPGs. This method allows the existing BT-WPGs to remain unchanged when the evicted cache line does not contain any branch target. Fig 3 (c), 1-*Target scheme*, adds two more fields in each cache line. *Instruction Pointer* is to record which branch instruction points to this cache line, so that only that BT-WPG needs to be invalidated. *Multiple bits* indicate whether multiple branches are created to this cache line. This field is implemented with a 2-bit saturated counter. If the higher bit is set to one, it means that there are multiple branches created to this cache line. In this case, the value is kept unchanged; otherwise the multiple field increases as long as a branch transfers to this line. When an I-Cache miss occurs, the multiple bits are checked. If it equals zero, no invalidation should be performed. If it is one, only the BT-WPG that the instruction pointer indicates will be invalidated, or all the BT-WPGs must be invalidated. Operations of the 1-T Scheme can be easily hidden into the cache miss process, while General, 0-T Scheme depends on the memory hierarchy and the implementation of the BTB extended array. Normally, when L1-Cache miss occurs, a single issue and in-order execution processor suffers at least ten cycle penalties. Invalidating the BT-WPGs in ten cycles can be realized by some RAM structures, or there are several cycle penalties during the total BT-WPGs invalidations. In Section IV, we do experiments and analysis that help choose a suitable invalidation scheme.

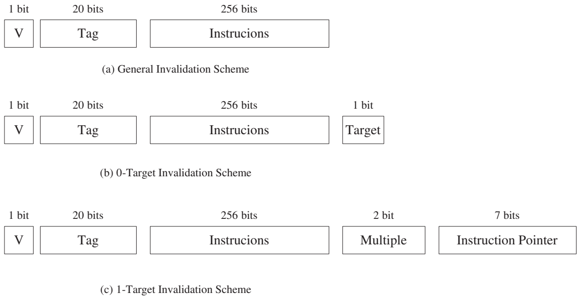


Fig. 3. The Cache Line Configurations of the WHOLE-Cache for the Invalidation Schemes

D. Performance Impact

First of all, the number of executed cycles does not increase. Our scheme does not perform the speculative operation. The way predicting cache requires one more cycle to search the other remaining ways when on a way miss prediction [6], [9]. However, the WHOLE-Cache, a tag comparison re-use approach, just behaves as accessing a conventional cache without any delay in that case.

Storing the way history into the I-Cache and the BTB respectively also makes no read or write conflict occur in a structure at the same time. The instruction cache of existing processor is organized using an individual array for valid bits, so the cache line invalidation could only accesses the valid-bit array [13]. Thus, by the same token, the additional memorization bits could be stored in a table accessed by the set-index address [6]. Therefore, the extension to the I-Cache and the BTB could be implemented by individual arrays. This structure could guarantee that the cache sub-bank which is being accessed can simultaneously be used for instruction fetch when a SL-WPG is updated.

Beyond the benefit brought by the individual array, the WHOLE-Cache could eliminate the potential possibility of cycles increasing when updating the BT-WPGs. Unlike [5], the WHOLE-Cache eliminates the write conflict possibility caused by updating the WPGs for a non-sequential flow. In the cycle of BTB updating, storing the tag comparison result of the target instruction into the BT-WPG and setting the target bit of cache line within the target instruction could be performed simultaneously, because the BT-WPG and the target bit are resided in different structures. Compared with the HBTC cache exploiting the BTB to record non-taken branches [7], there is no not-taken branch entry which needs to be updated in the WHOLE-Cache.

All extra operations can be hidden into the conventional operations in the WHOLE-Cache. When a taken branch updates its BTB entry in the WHOLE-Cache design, its related BT-WPG could also be updated at the same time. Thus, the updating of a BT-WPG for a non-sequential flow could be hidden in a conventional BTB update operation,

TABLE I

THE CONFIGURATION OF OUR BASELINE PROCESSOR

Component	Description
CPU	In-order, 1-issue, 32-RUU size
Branch Prediction	BTB 128 entries, direct mapped bimodal 128 entries, using pre-decoding 5 cycles branch penalty
L1 I-Cache	32KB, 4-way, 32 byte block, 2cycle
L1 D-Cache	32KB, 4-way, 32 byte block, 2cycle

so it will not increase the frequency of accessing BTB in the WHOLE-Cache. Because of the respective extension structure, the operation of invalidating the SL-WPG and the BT-WPG could also be performed simultaneously. Therefore, this does not increase the process cycles of the cache miss either.

Secondly, whether the WHOLE-Cache influence the processor frequency depends on the processor implementation. There are five more judgments compared with a conventional set-associative cache. However, those judgements are quite simple: whether the instruction is the last instruction in a cache line, whether the instruction is the first instruction in a cache line, whether the SL-WPG is valid, whether the BT-WPG is valid and the access mode selection. The first two judgments can be performed with the I-Cache access simultaneously. And since accessing the SL-WPG or BT-WPG can be hidden into the conventional I-Cache or BTB access, the two judgments of valid WPG do not increase the cycle time either. Only the Access Mode Selector lengthens the datapath of instruction fetch. If the instruction fetch datapath is the worst path of the processor, this would impact the clock frequency, or the WHOLE-Cache design has no influence on the clock.

IV. PERFORMANCE AND ENERGY EVALUATION

In this section, we evaluate the performance and energy efficiency of the WHOLE-Cache by using SimpleScalar simulator [1] with Wattch extensions [2]. The basic parameters were derived using TSMS 65nm technology (worst case) from a WHOLE-Cache Verilog Model.

Table I describes the details of the baseline single issue and in-order execution processor used in each of the experiments. The WHOLE-Cache was simulated on some programs, which are representative of fairly complex general purpose applications: we used four programs of the SPEC CPU 2000 benchmark suite (164.*gzip*, 176.*gcc*, 171.*swim*, 172.*mgrid* using ref input) [17] with selected simulation points of 100 million instructions using a methodology based on SimPoint [20]. And three Mediabench programs (*adpcm*, *jpeg*, *mpeg*2) are also evaluated, each with a decoder and an encoder [16].

A. Discussion on the Implementation Issues

The modification from a conventional I-Cache to the WHOLE-Cache is simple to implement as follows. 1) CWP is added in the Instruction Fetch Unit. 2) An input port of the WPG should be augmented to the I-Cache structure, which is directly connected to the CWP. The WPG port only consists of 2-bit way pointer and 1-bit valid flag. 3) *Fetch Directly State* is added to the Finite State Machine (*FSM*) of the I-Cache. When the input valid flag is set, the FSM transfers to this state and only reads one SRAM for the output. 4) The WPG arrays related with the I-Cache and the BTB are just SRAM structures with single port controlled by the additional control logic. 5) The Access Mode Selector.

TABLE II

PARAMETERS DERIVED FROM TSMS 65NM TECHNOLOGY (FREQUENCY 1GHZ, DRIVING BUFX4)

TSMC 65nm Tech (Worst Case)	Timing			Power	
	Worst Timing Path	SRAM: clk -> Q		Dynamic Power (mw)	Leakage Power (mw)
		1024x64	512x24		
4-Way ICache	1.20	1.11	0.97	106.6	2.79
BTB	1.09	0.90	0.93	6.16	0.133

We have modeled the WHOLE-Cache design by modifying the I-Cache of a single issue and in-order execution commercial processor. This RISC processor has 8-stage pipeline and 32KB 4-way set-associative L1-I/DCache. Under TSMC 130nm technology, it runs at a maximum of 675MHz. Then we used this model for deriving the parameters of the TSMC 65nm technology evaluation. Some parameters of the WHOLE-Cache model are stated in Table II. Since the Access Mode Selector does not lie in the worst case path of the processor, the cycle time does not increase.

B. Energy Consumption Model

The total energy dissipated in the WHOLE-Cache ($E_{WHOLE-Cache}$) can be expressed as follows:

$$E_{WHOLE-Cache} = E_{ICache} + E_{BTB} + E_{ICache\_add} + E_{BTB\_add} + E_{Logic} \tag{1}$$

where E_{ICache} and E_{BTB} are the energy consumed in the instruction cache and in the BTB, and E_{ICache_add} and E_{BTB_add} are the energy for I-Cache and BTB extension respectively. E_{ICache} can be approximated to the total energy consumption for reading tags or data lines. The E_{ICache_add} can be approximated to the energy for accessing and invalidating the SL-WPG array. $E_{BTB} + E_{BTB\_add}$ depends on the number of each operation in BTB and the summary energy consumed by reading, writing and invalidating the BT-WPGs, including the energy consumption by the feedback path from cache to the core. E_{Logic} is the energy dissipated by the WHOLE-Cache peripheral logic. According to the analysis in section III, the control logic of instruction fetch selection mode and invalidation scheme (i.e. 0-T scheme) totally only requires two 5-bit and four 1-bit comparators, four 1-bit registers and nine related muxipliers, so we do not take the energy consumed by the control logic of the WHOLE-Cache (E_{logic}) into consideration. Especially, if the indexing of the I-Cache is accomplished using bits from the page offset,

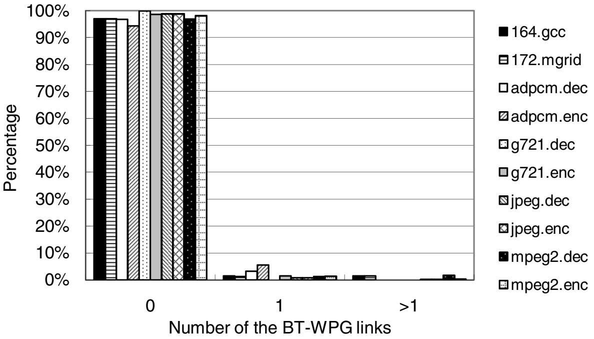


Fig. 4. Distribution of the number of BTB entries that need to be invalidated on each cache line eviction

besides the elimination of tag comparisons, the WHOLE-Cache could also avoid the ITLB access when the target instruction is guaranteed to be in the I-Cache. This is because the virtual to physical address translation does not affect the page offset portion of a fetch address [10]. According to our baseline processor, when fetching instruction directly, the ITLB access could also be eliminated.

C. Invalidation Scheme Selection

Fig. 4 shows the distribution of the number of BTB entries that need to be invalidated on each cache line eviction running a subset of benchmarks. (Since the number of replaced cache line of some programs is quite small, their data are not collected, such as 164.gzip and 171.swim.) It shows at least 94% of the evicted cache lines among the selected programs have no need to invalidate a BTB entry. Because the WHOLE-Cache only makes the links to a cache line for the taken branches, which is different from [5]'s using total branches, the probability of invalidation of the total BT-WPGs in the WHOLE-Cache is quite low. Thus, 0-Target Scheme is preferred to the General Scheme or the 1-Target Scheme, which uses more 8K bits but most are redundant. From the same experiment, less than 1% of the total number of the valid used WPGs are invalidated by the Conservative Scheme. We suggest using Conservative Scheme for simple implementation. Therefore, Conservative+0-T Scheme is applied in the following evaluations.

D. Performance/Energy Efficiency

Table III describes the structures of different mechanisms compared with the WHOLE-Cache and also shows the approximate values of their hardware resource requirements. This requirement of the WHOLE-Cache is quite smaller than the way memorization design, which needs 24K bits. Compared with [10], the WHOLE-Cache needs almost the same quantity as a TL_16x4 TH-IC, but much smaller resource than a TL_32x4 configuration.

TABLE III
THE STRUCTURE DESCRIPTION OF DIFFERENT MECHANISMS

Mechanism	Description	Resource
Way Prediction	1024 Entries	2K
Way Memorization	Each cache line is augmented with a sequential link and every other word has a branch link Using Zero-link Scheme	24K Bits
HBTC Cache	Direct mapped 128 entries BTB records both taken and not-taken branch with interline tag comparison scheme (ITC)	4K Bits
WHOLE-Cache Conservative 0-T	Each cache line has a SL-WPG and a Target bit. Extends each BTB entry with a BT-WPG	4K Bits

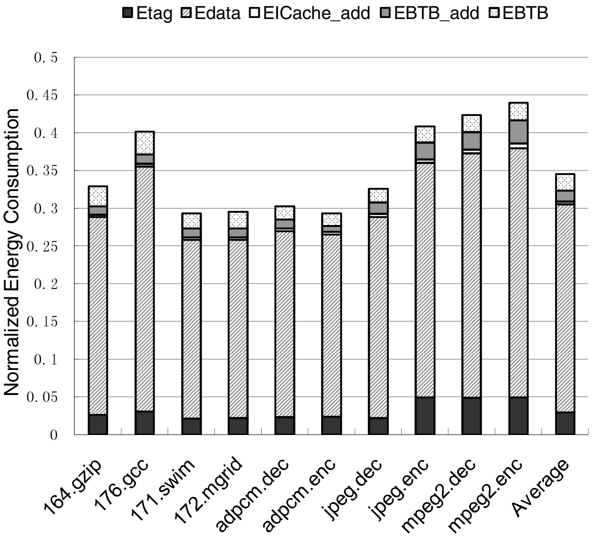


Fig. 5. Energy Consumption of the WHOLE-Cache and Its Breakdown

Fig. 5 shows the energy consumption of each program required by the instruction fetch stage using the WHOLE-Cache and its breakdown. The results are normalized to the value of the baseline processor configuration. From the evaluation, the average fetch power is approximately 34.88% of the baseline fetch power. The maximum saving was 70.15% achieved for the 171.swim program. It is observed that the energy consumption of the additional bits for the way history storage is quite small compared with the dynamic energy reduction of reading the I-Cache data array. Therefore, although the WHOLE-Cache augments extension bits in the BTB and the I-Cache, the significant effectiveness of the energy reduction has compensated for these costs. The fetch energy reduction comes from the omission of unnecessary way activation and tag comparison.

Figs. 6 and 7 show the comparison of energy reduction and performance between several existing approaches and the WHOLE-Cache. We conclude that the WHOLE-Cache performs best in every program: it had the greatest energy reduction and no performance penalty occurred. Although way prediction occupies the least resources, it still needs to read one tag to verify the prediction and the extra readings compensated for the wrong prediction, which reduces the effect of energy reduction and the IPC. The two problems of the way memorization are the lots of large quantity of redundant extra resources and the presence of write conflicts in the same structure. Recording the not-taken BTB makes it less competitive than the WHOLE-Cache.

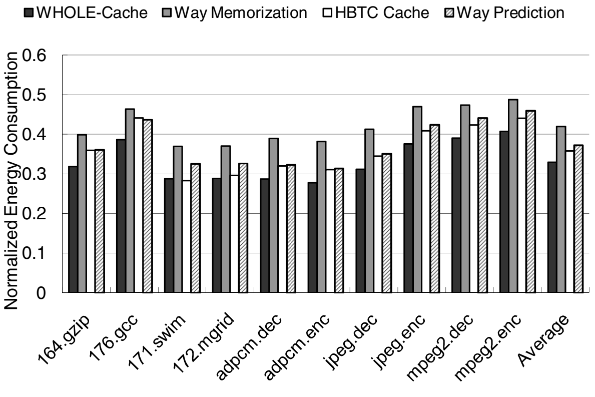


Fig. 6. Energy Consumption of Different Approaches

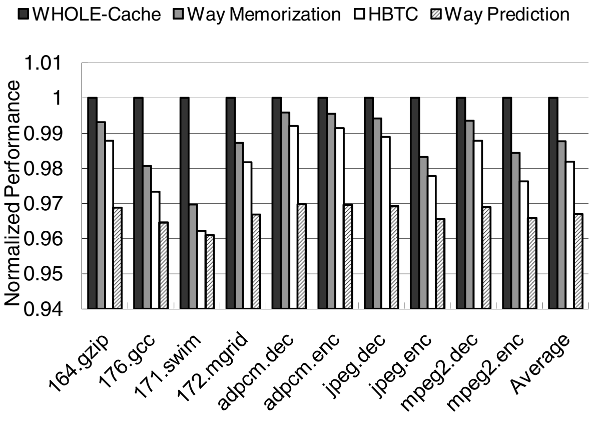


Fig. 7. Performance Evaluation of Different Approaches

V. CONCLUSION

In this paper, we have proposed a way history oriented low energy instruction cache design for single issue and in-order execution processors. By exploiting the instruction fetching behavior, the WHOLE-Cache develops the I-Cache and BTB (RAS) respectively to record the way history for the direct instruction fetching in the set-associative cache based on their original structural features. The WHOLE-Cache performs direct fetch to the cache access without activating unnecessary way activation, tag lookups or even ITLB accesses. The WHOLE-Cache not only achieves the dynamic energy reduction and less redundant bits, but also leads to additional cycle penalties because of the operation of hiding, conflict elimination mechanism and simple implementation.

We evaluate the effectiveness of this technique in reducing total instruction fetch energy by using the parameters of TSMC 65nm technology to our baseline processor. The WHOLE-Cache achieves 69.30% energy reduction on average for the 4-way instruction cache, while the total energy reduction of the instruction fetch stage is up to 65.12% without any delay.

We also evaluated the scalability of the WHOLE-Cache. In a case where the cache associativity is 2, 4, 8 and 16, the average energy are 55.2%, 34.8%, 24.3%, 18.6%, respectively. However, the additional bits for storing tag comparison results consumes more energy with the cache associativity increases and the effect of energy dissipated by the WHOLE-Cache is reduced. We leave this optimization to future work.

VI. ACKNOWLEDGMENTS

We would like to thank Li Xian Feng, Lu Junlin of MPRC for their help with the paper review, and Shang Zheng, Shi Yunfeng for the help with the cache model.

REFERENCES

[1] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35:59-67, February 2002.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual International Symposium on Computer architecture*, pages 83-94, New York, NY, USA, 2000. ACM Press.

[3] 3rd Generation Intel XScale Microarchitecture Developer's Manual, May 2007, *http://developer.intel.com*

[4] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proceedings of the 1997 International Symposium on Microarchitecture*, pages 184-193, 1997.

[5] A. Ma, M. Zhang, and K. Asanovic. Way memorization to reduce fetch energy in instruction caches. *ISCA Workshop on Complexity Effective Design*, July 2001.

[6] K. Inoue, T. Ishihara, and K. Murakami. Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption. In *Proceeding of International Symposium on Low Power Electronics and Design*, pages 273-276, August 1999.

[7] K. Inoue, V. Moshnyaga, and K. Murakami. A Low-Energy Set-Associative I-Cache with Extended BTB. In *Proceeding of International Conference on Computer Design*, pages 148-153, September 2002.

[8] M.Muller. Power Efficiency & low cost: The ARM6 family. In *Hot Chip IV*, August 1992.

[9] M. D. Powell, A. Agarwal, T. N. Vijaykumar, and B. Falsafi. Reducing set-associative cache energy via way-prediction and selective direct mapping. In *Proceedings of the 34th International Symposium on Microarchitecture (MICRO 34)*, pages 54-65, Dec. 2001.

[10] S. Hines, D. Whalley, and G. Tyson. Guaranteeing Hits to Improve the Efficiency of a Small instruction Cache. In *Proceedings of the ACM SIGMICRO International Symposium on Microarchitecture*, pp. 433-444 December 2007.

[11] J. Montanaro, et al. A 160-mhz, 32-b, 0.5-W CMOS RISC microprocessor. *Digital Tech. J.*, 9(1):49-62, 1997.

[12] D. Parikh, K. Shadron, Y. Zhang, and M. Stan,. Power-aware branch prediction: characterization and design. *IEEE Transaction on Computers*, vol. 53, no. 2, pp. 168-186, Feb. 2004.

[13] OpenSPARC T1 Microarchitecture Specification, *Sun Microsystems*, Aug 2006, pp. 9

[14] Krste Asanovic, et al. The landscape of Parallel Computing Reserach: A view from Berkeley. *Technical Report No. UCB/EECS-2006-183*, December 18, 2006.

[15] J Kin, M Gupta, WH Mangione-Smith. Filtering memory references to increase energy efficiency. *IEEE Transactions on Computers*,49(1): 1-15, 2000

[16] C Lee, M Potkonjak, WH Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. *The University of California at Los Angeles*

[17] J. Henning. SPEC2000: measuring CPU performance in the new millennium. *IEEE Computer*, July 2000.

[18] Timothy M.Jones, Sandro Bartolini, Bruno De Bus, John Cavazos and Michael F.P.O'Boyle. Instruction Cache Energy Saving Through Compiler Way-Placement. *Design, Automation and Test in Europe*, March 2008.

[19] N.S. Kim, K. Flautner, D.Blaauw, and T. Mudge. Single-Vdd and single VT super-drowsy techniques for low-leakage high-performance instruction caches. *Proceeding of International Symposium on Low Power Electronics and Design*, pp.54-57, 2004.

[20] E.Perelman, G.Hamerly, M.Van Biesbrouck, T.Sherwood, and B.Calder. Using SimPoint for Accurate and Efficient Simulation. *ACM SIGMETRICS*, June 2003.