IEICE TRANS. INF. & SYST., VOL.E94–D, NO.7 JULY 2011

PAPER

Analysis before Starting an Access: A New Power-Efficient Instruction Fetch Mechanism

Jiongyao YE[†a)], ***Member***, **Yingtao HU**[†], ***Student Member***, **Hongfeng DING**[†], ***Nonmember***, ***and*** **Takahiro WATANABE**[†], ***Member***

SUMMARY Power consumption has become an increasing concern in high performance microprocessor design. Especially, Instruction Cache (I-Cache) contributes a large portion of the total power consumption in a microprocessor, since it is a complex unit and is accessed very frequently. Several studies on low-power design have been presented for the power-efficient cache design. However, these techniques usually suffer from the restrictions in the traditional Instruction Fetch Unit (IFU) architectures where the fetch address needs to be sent to I-Cache once it is available. Therefore, work to reduce the power consumption is limited after the address generation and before starting an access. In this paper, we present a new power-aware IFU architecture, named Analysis Before Starting an Access (ABSA), which aims at maximizing the power efficiency of the low-power designs by eliminating the restrictions on those low-power designs of the traditional IFU. To achieve this goal, ABSA reorganizes the IFU pipeline and carefully assigns tasks for each stages so that sufficient time and information can be provided for the low-power techniques to maximize the power efficiency before starting an access. The proposed design is fully scalable and its cost is low. Compared to a conventional IFU design, simulation results show that ABSA saves about 30.3% fetch power consumption, on average. I-Cache employed by ABSA reduces both static and dynamic power consumptions about 85.63% and 66.92%, respectively. Meanwhile the performance degradation is only about 0.97%.

key words: *low power, instruction cache, instruction fetch mechanism*

1. Introduction

Power efficiency is important to modern microprocessor applications (e.g. notebook computers, consumer electronics and cellular phones). They require not only high performance, but also low power consumption for longer battery life. Another driving force behind designing for power efficiency is that power consumption is becoming the limiting factor in integrating more transistors on a single processor or on a multi-processor module due to the cooling, packaging and reliability problems. Especially, I-Cache as the important component of IFU usually dissipates a considerable portion of power in modern processors. For example, the on-chip caches of the 211164 DEC Alpha chip dissipate 25% of the total power of the processor [1]. The Strong ARM SA-110 processor from DEC, which targets specifically low-power applications, dissipates about 27% of the power in I-Cache [2]. In the Pentium Pro processor, the IFU and the I-Cache contribute 14% to the total power consumed [3]. Thus optimizing the power consumption of I-Cache is particularly important.

Up to now, many low-power techniques on I-Cache have been proposed to achieve a power-efficient IFU. In general, these techniques save power by analyzing the fetch address to avoid some unnecessary accesses such as accessing only the predicted cache way instead of all of the way. However, in the traditional IFU, I-cache needs to be immediately accessed once the fetch address is available. This leads to the limited work which can be done by the power-saving techniques after fetch address generation and before starting an access. Therefore, the existing low-power techniques usually lose possibility of maximizing power efficiency, or make it necessary to increase the access delay and design complexity. This problem stimulates us to seek a better power-efficient IFU architecture.

In this paper, a new power-efficient IFU architecture, Analysis Before Starting an Access (ABSA), is proposed to maximize the efficiency of the low-power design. In ABSA, before starting an access, a separate stage is introduced to get useful information (*i.e.*, way or subbank information) for the low-power strategies by analyzing the fetch address or other resources such as Branch Target Buffer (BTB). Then, utilizing the analysis results, the supply voltage of the required I-Cache line is awakened in the subsequent pipeline stage. At last, the instructions are fetched in the last stage of IFU. As a result, the power consumption of I-Cache can be reduced more effectively by performing more careful strategy before starting an access. Furthermore, ABSA provides a good low-power design space not only for I-Cache but also for other power consumers of IFU (*e.g.*, branch prediction). Our proposed ABSA can maximize the power efficiency of the low-power design in IFU by providing sufficient time and information without significant performance overhead and design complexity.

The remainder of this paper is organized as follows. Section 2 discusses the related works and analyzes traditional IFU architectures. Next, the ABSA design is presented in Sect. 3. We describe the ABSA-based low-power implementation in Sect. 4. The performance and scalability of ABSA is discussed in Sect. 5. We show the experimental results in Sect. 6. In Sect. 7, we conclude this paper.

2. Related Works

Several researchers have worked on reducing power con-

Manuscript received November 10, 2010.
Manuscript revised February 28, 2011.

[†]The authors are with the Graduate School of Information, Productions and Systems, Waseda University, Kitakyushu-shi, 808–0135 Japan.
a) E-mail: yea_asgard@suou.waseda.jp

DOI: 10.1587/transinf.E94.D.1398

sumption of the cache. Filter cache [4], L-Cache [5], block buffer [6] and multiple line buffers [7] employ a small storage unit between the processor and the level one cache to avoid unnecessary cache lookups. Cache subbanking was proposed by Su [6] to reduce power consumption in caches by fetching only the requested subline, rather than the entire logical cache line. Further study by Ghose and Kamble [7] divides the data array not only vertically but also horizontally into several segments of bitcells to get more power savings. In this technique, greater power reductions are achieved with less precharge drivers and sense amplifiers. Way-prediction [10] is proposed to reduce the power consumption of the set-associative cache, which saves power by first accessing only the predicted cache way. It accesses other ways only when the prediction is incorrect. This approach highly depends on the way-prediction accuracy, and causes indefinite cache hit time. A better approach is the two-level filter scheme [11], which accesses a block buffer and sentry-tag arrays ahead of cache data. The block buffer eliminates the unnecessary cache accesses, and the sentry-tag further filters out the unnecessary way activities in case of the block buffer miss. It reduces the total cache power consumption of the 32 KB two-way set-associative I-Cache by about 56.79%.

Other techniques focus on leakage power consumption (e.g. drowsy caches [12], cache decay [17], gated-V_{DD} [18]). Especially, Drowsy I-Cache [13], a representative state-preserving technique, employs dynamic voltage scaling (DVS) circuit technique and subbank prediction scheme to selectively wake up one I-Cache subbank and keep other in the low-voltage mode. It reduces leakage power consumption in I-Cache by about 75%.

Although these techniques can reduce power consumption of I-Cache, we find that their efficiencies are still restricted by the traditional IFU architectures. In general, before starting an access, those techniques in traditional IFU architectures usually require some extra time or complex hardware support to identify and eliminate unnecessary accesses. Two basic IFU architectures are widely used in modern processor design: one is the branch predictor is accessed after I-Cache [16], which results in a one-cycle branch delay (e.g. Alpha21164 [22], Intel i960 [23]). Another, such as ultraSPARC-III [19], is proposed to access branch predictor in parallel with I-Cache to reduce the branch delay. And, most current advanced processor cores, such as IBM Power 6 [20] and Pentuim 4 [21], basically follow this IFU architecture design. A common characteristic of both traditional IFU architectures is that the fetch address needs to be sent to the I-Cache as soon as it is generated. Therefore, spare time and information to eliminate the unnecessary access are quite limited. For example, Drowsy I-Cache with the horizontally configuration accesses the subbanks in all cache way due to lack of cache way selection. To reduce the access delay, the two-level filter cache must accesses the row decoder of each way while accessing the L1 and L2 filter.

Due to such restrictions of the low-power designs in the traditional IFU are prevented from maximizing power efficiency. Thus, to use more time and information to identify and eliminate unnecessary accesses, the traditional processors cannot but increase the access delay or reduce the core frequency or increase the design complexity. To achieve a better tradeoff between performance and power, a new IFU architecture that is more suitable for another smart low-power approach is indispensable.

3. ABSA Design

Based on the observations on the traditional IFU architectures and the previous low-power approaches, we propose a new power-efficient IFU where the low-power techniques reduce the power dissipation more effectively. Our design methodology is based on the following major principles:

1. The primary aim of our work is to effectively reduce power consumption on IFU at the cost of minimal performance overhead. Our proposed IFU with the deeper pipeline technique reorganizes the pipeline depth of IFU and carefully assigns the tasks for each stage, which makes ensure that the frequency and bandwidth can not be reduced. Meanwhile, the deeply pipelined IFU can provides sufficient time and information for the low-power strategies to maximize the power efficiency. However, this approach incurs larger branch misprediction penalties. In Sect. 5.1, two approaches are discussed to alleviate the branch misprediction penalties in our design.
2. The low-power strategies used in our paper must be carefully selected. The prediction-based low-power strategies, such as the way-prediction techniques, are not suitable for our purpose. It is because the performance, access delay and power consumption depend on their prediction accuracy. Our proposed IFU should be capable of providing the sufficient time and information for the low-power techniques, so that analysis-based approaches are used to save more power due to more accurate access because they can avoid unnecessary access by analyzing the fetch address or other information before starting an access
3. The scalability is also an expected target for our proposed IFU design. Our design can still maintain power efficiency even if the fetch bandwidth and core frequency increase. Moreover, the power-efficient IFU is designed not only for I-Cache, but also for other power consumption contributors of IFU.

3.1 ABSA Architecture

According to the above design methodology, ABSA is composed of the basic four stages as shown in Fig. 1. Like as other general processors, in the first stage (s0), the fetch address is calculated either by incrementing the previous address or by selecting a new address in response to a predicted or actual flow change operation. An individual stage, named Analysis stage (s1), is added prior to the I-Cache ac-

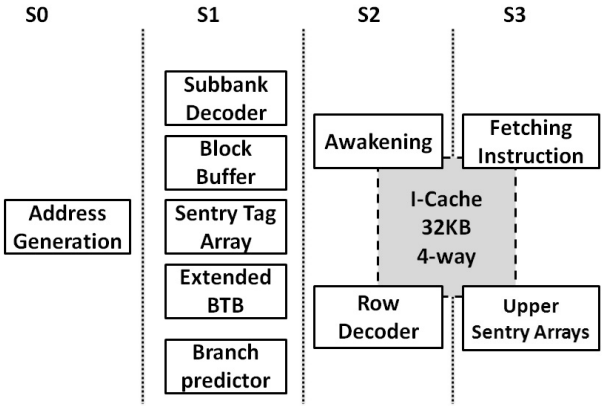

Fig. 1 ABSA architecture.

cess, which provides useful information for the low-power techniques to filter unnecessary I-Cache access. Moreover, this stage can also generate the control information (e.g. way selection and subbank selection) by analyzing the fetch address for the subsequent I-Cache access when I-Cache access cannot be avoided. Then, instruction fetch process can implement a more accurate selection to maximize the power efficiency in the stages 2 and 3. The I-Cache access is distributed over two stages: the Wakeup stage (s2) and the Fetch stage (s3). In the wakeup stage, Dynamic Voltage Scaling (DVS) technique is employed for leakage reduction. Only the required subbank needs to change its supply voltage to normal V_{DD}, others maintain the drowsy mode. Finally, the instructions can be accurately accessed according to the information provided by the analysis stage. The tasks of each stage are introduced in the following sections.

3.2 Analysis Stage

Different from the traditional IFU pipeline stage, the analysis stage is newly added. Various function blocks are used to analyze the fetch address in order to filter the unnecessary access and provide the information for I-Cache access. As illustrated in Fig. 1, the analysis stage (s1) includes a subbank decoder, a block buffer, a sentry tag array, an extended BTB and a branch predictor.

The subbank decoder determines which subbank and its row decoder are activated. Low order index bits are fed to the subbank decoder to do this selection. In the traditional IFU, this proposed logic would increase the cache access time since it needs to be accessed in series with the row decoder. In ABSA, the subbank decoder is one stage ahead of the row decoder, so that the pipelined decode operation would hide the cache access delay caused by the serial decode operation.

A block buffer [6] is used for filtering accesses to the whole I-Cache. If the block buffer contains the instructions to be fetched, the access to I-Cache is avoided. In addition, the tag array of I-Cache is divided into two parts. The first part, named Sentry tag array [11], [27], contains lower bits of the tags to decide which cache way needs to be accessed. The second part, named Upper-Tag array, contains the remaining tag bits. The sentry-tag array is located two stages ahead of the upper-tag array and fetching instructions. The benefit is that the access for each cache way, including of its row decode and precharge, is avoided if the sentry-tag array identifies a miss according to the comparison result between a sentry-tag and a fetch address in the analysis stage. Furthermore, using the sentry-tag array does not lead to performance degradation caused by way misprediction.

An extended BTB can maintain not only the target address but also some other useful information for the low-power techniques. The detailed design of the extended BTB is introduced in Sect. 4.1. As same as ultraSPARC-III, the *gshare* branch prediction is employed, and is performed in the analysis stage because it is not too complex. Although the main concern of our work is power-saving on I-Cache since it is the biggest power consumption contributor of IFU, other function blocks for more detailed analysis can be also employed in the analysis stage to bring higher power saving.

3.3 Wakeup and Fetch Stages

In the wakeup stage (s2), according to the result of the subbank decoding and way selection, only one row decoder is activated to decide which cache set needs to be accessed. Concurrently, the supply voltage of all lines in the selected subbank is changed to the normal V_{DD} while other subbanks are in drowsy mode. Different from the DVS technique used in drowsy I-Cache, the control signal consists of the subbank selection signal and way selection signal, which ensure only the subbank in the selected cache way is active at a time. Finally, the remainder tag comparison is performed and the fetch instructions are accesses in the fetch stage.

4. ABSA-Based Low-Power Implementation

In this section, we will first describe the extended BTB and the power-efficient I-Cache configuration in order to support our proposed IFU. Then, we explain ABSA how to reduce the power consumption of I-Cache.

4.1 Extended BTB

The extended BTB is intended to provide more information about the branch instructions in the analysis stage. As shown in Fig. 2, the extended BTB includes not only three traditional fields (*i.e.*, the valid bit, Branch instruction Address (BA) and Branch Target Address (BTA), but also two new extra fields: Change Page Field (CPF) and Sentry Tag Field (STF). CPF is only one bit, which records whether the branch instruction will change to another physical page when it is taken. The sentry tag of the branch target address is recorded in STF. It can be directly compared to the sentry tag array to decide which cache way is valid when the branch is taken.

When a branch is resolved, the extended BTB needs to be updated only when it misses for this branch. However,

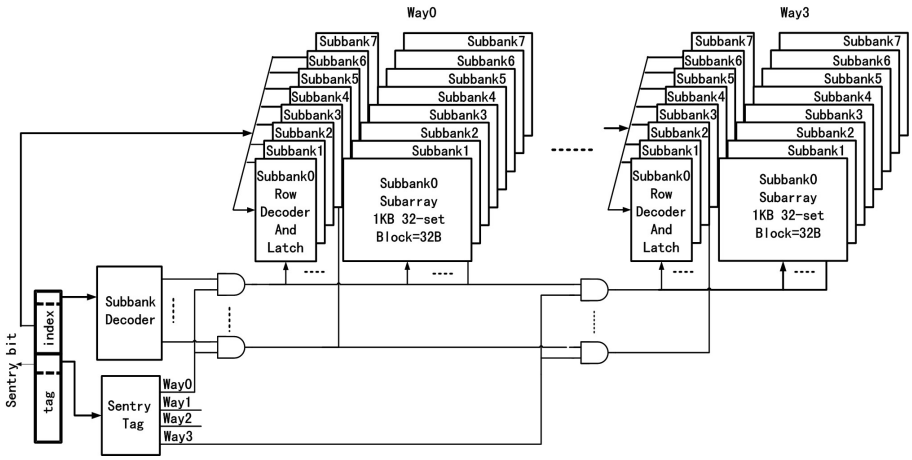

Fig. 3 Power-efficient I-Cache architecture.

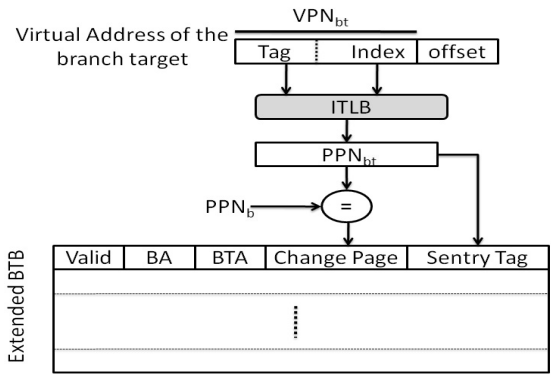

Fig. 2 Extended BTB architecture.

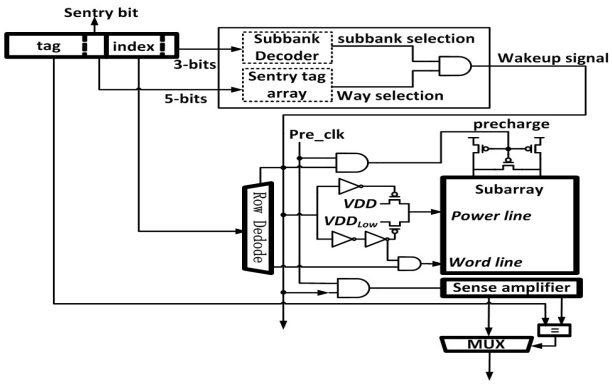

Fig. 4 DVS Implementation of the I-Cache line.

VPN$_{bt}$: Virtual Page Number of branch target
PPN$_{bt}$: Physical Page Number of branch target
VPN$_b$: physical Page Number of branch instruction
ITLB: Instruction Translation Lookaside Buffer
BTA: Branch Target Address
BA: Branch instruction Address

the sentry tag cannot directly be extracted from the branch target address because a virtual memory system [29] is used in our design, in other words, the branch target address is not a physical address but a virtual address. Thus, Instruction Translation Lookaside Buffer (ITLB) is accessed by the Virtual Page Number of branch target (VPN$_{bt}$) and the Physical Page Number of the branch target (PPN$_{bt}$) is found. As illustrated in Fig. 2, VPN$_{bt}$ is translated into PPN$_{bt}$ to generate both STF and CPF. The sentry tag is the corresponding bits of PPN$_{bt}$ to be saved in STF, and the comparison result between the Physical Page Number of the branch instruction (PPN$_b$) and PPN$_{bt}$ is saved in CPF.

4.2 Power-Efficient I-Cache Configuration

Considering performance and compatibility, a power-efficient I-Cache is modeled in our simulation, which stems from the principle that a large cache is broken down into smaller block for both performance and power. And, it need not be supported by software. As shown in Fig. 3, the 32 KB 4-way set-associative I-Cache is partitioned into eight 4 KB subbanks in the horizontal configuration. The size of each subbank is distributed through all cache ways so each portion of the subbank in one cache way, called subarray, is 1 KB (set = 32 and block = 32 B). Figure 4 illustrates the DVS circuit technique for each cache line. A separate modified precharge circuit is provided for each subarray. For each cache line, the wakeup signal is decided by the comparison results of the sentry tag and the subbank decoder. In our paper, when a cache line will be accessed, only 1 KB subarray is woken up by a normal V_{DD}. Different form the drowsy I-Cache [13] in the vertically scheme, the horizontal scheme will not cause the performance degradation that is caused by the fetch instructions saved in an inactive cache way. Furthermore, the traditional horizontal scheme is not power-efficient since all the cache ways are accessed at a time. In ABSA, the analysis stage provides not only subbank selection signal but also way selection signal. Thus, more power saving can be accomplished because only one portion of the subbank of the selected cache way is active.

4.3 Reducing I-Cache Power

I-Cache used in our design exploits the fact that a large cache is broken down into smaller subarrays to reduce the wiring and diffusion capacitances of bit lines as well as the wiring and gate capacitances of word lines used to activate the memory cells. The reduced capacitance helps both the cache access time and dynamic energy consumption lower when accessing the caches. And, smaller drivers, precharging transistors and sense amplify can be used while partitioning the data array horizontally into several segments of bitcells.

ABSA activates only one portion of the subbank in the selected cache way by the wakeup signal provided by the analysis stage, and keep other subarray in the low-voltage mode. Then, the dynamic and static power consumption can be reduced. In this way, only one subarray can be row-decoded, precharged and accessed in the wakeup and the fetch stage, respectively. Figure 5 illustrates the detailed process of wakeup signal. The subbank selection is performed by subbank decoder. The three low order bits of the index are fed to the subbank decoder to select one of eight subbanks. For way selection, the two-level filter technique is used in the analysis stage. First, the block buffer is accessed to check if it has contained the fetch instructions. At the same time, the sentry tag array indexed by the fetch address in each way is compared with the sentry bit of the fetch address. Only when the block buffer misses and the sentry tag hits, the corresponding subarray can be accessed.

However, as the same as the above mentioned, the current virtual address cannot directly generate the sentry bit for comparison. To address this problem, the last result of ITLB lookup is saved into a special register called the Last Physical Page Number (LPPN) that includes the VPN and its corresponding PPN of the last ITLB lookup. The current virtual address generated by the fetch address is directly compared with the VPN in the LPPN. If they match, then the current instruction is in the same page as the last one, so the PPN in the LPPN as the current PPN can be compared with the sentry tag array. This approach is based on the tendency of spatial locality that the dynamic instruction sequence tends to sequentially increasing in the same page. There are two ways by which a program execution can move from one instruction page to another: 1) the branch target may not be in the same page as the branch instruction if the branch predictor predicts taken and extended BTB provides the target address (we call this the **branch case**), and 2) two successive instructions which are on page boundaries (we refer to this as the **boundary case**). In branch case, the sentry tag field of extended BTB as the current PPN can be compared with the sentry tag array. However, in the boundary case, the PPN in the LPPN cannot be used for sentry comparison, and the wakeup signals of all subarrays of the selected subbank are valid.

As shown in Fig. 5, the current sentry tag maybe come from LPPN or STF of the extended BTB. An 1-bit control signal, called Last Branch Instruction Recorder (LBIR), is used to decide which sentry tag source is valid. It is set to zero if the current fetch address is the branch target address that is provided by BTB at the last cycle. Otherwise, it is set to one. Note that the I-Cache precharge and access can be generally avoided if the sentry tag comparison result is not equal. But, in the boundary case, the current VPN does not fit on the VPN in the LPPN (mismatch output is 1), and LBIR is set to one, OR logic gate (G1) masks the way selection signal from the sentry tag comparison. The result is that the all cache ways of the selected subbank need to be accessed. Fortunately, since the change of virtual page number is not frequent in the continuous instruction stream, the boundary case does not occur frequently. The results of our experiment show that the contributions of the boundary cases is less 3% during the whole program execution.

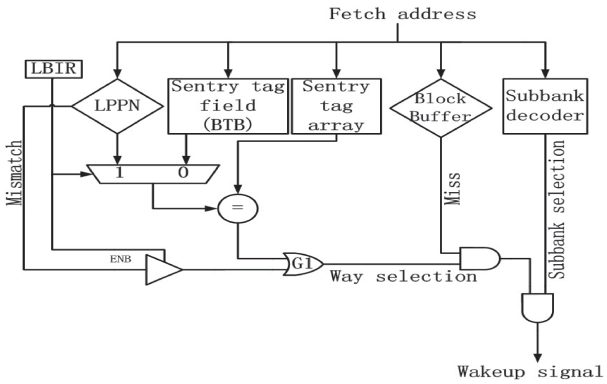

Fig. 5 Processing of wakeup signal.

LBIR: Last Branch Instruction Recorder
LPPN: Last Physical Page Number

5. The Analysis of Performance, Scalability and Area

In this section, we first investigate the delay penalty due to ABSA. Then, the scalability of ABSA is discussed to present the applicability of this new power-efficient IFU architecture. Finally, we also take the hardware cost of ABSA into concern.

5.1 Delay Penalty

ABSA adopts the deeper pipeline technique to reduce both of the cache access time and the power consumption. Two kinds of delay are removed from the critical path of I-Cache access. First, by moving sentry tag array to analysis stage, the sentry bits' comparison is eliminated from the critical path of I-Cache access. Second, the subbank decoder and the row decoder are moved to the analysis and the wakeup stages, respectively. Thus, the time to access the row decoders of I-Cache is hidden. The quantitative analysis of the critical path is presented in Sect. 6.3.

The most important contributor of performance degradation is branch misprediction penalty due to ABSA. Compared with the traditional IFU, two newly added stages (i.e.,

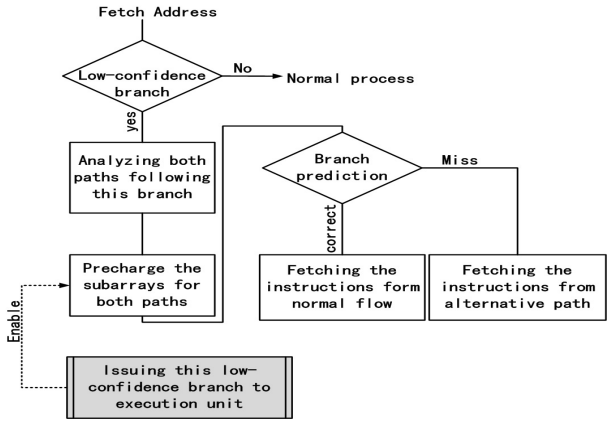

Fig. 6 ABSA_bal process flow.

the analysis and the wakeup stages) result in larger branch misprediction penalty. This adverse effect can be alleviated by two approaches. One is to improve the precision of the branch prediction, which is the most common and effective way to reduce the branch misprediction penalty in the deeper pipelining processors [24]. Second is a balanced method (named ***ABSA_bal*** method), which provides a tradeoff between power consumption and performance. The detailed process flow of ABSA_bal is shown in Fig. 6. A low-confidence mechanism [15] is employed in the analysis stage to identify the branches most likely to be mispredicted. Once such branch is encountered, the analysis stage allows analyzing the address of the instructions from the both paths following this low-confidence branch, simultaneously. And, the analysis result of the alternative path is temporarily saved in the analysis stage as the alternative wakeup signal. When this branch is issued into the execution unit, the subarray containing the instructions from the alternative path following this branch is pre-awaked by the alternative wakeup signal. Then, if the branch is resolved and misprediction is discovered, the instructions from the alternative path can be directly fetched in the fetch stage, which reduces the branch misprediction penalty. But, the static power consumption may increase because more than one subarray may be awaked during the execution period of the branch.

5.2 Scalability

A basic design principle of ABSA is that it can realize the full potential of future semiconductor processes as process technology evolves. Scalability is therefore of major importance. When the processor runs at a higher frequency and higher fetch bandwidth, ABSA is more scalable than the traditional IFU architectures.

With the continuous increment of a cache size and the array of the branch predictor, the design complexity of the low-power strategies increases so that more times for analysis and prediction are required before I-Cache is accessed. However, as the processor frequency grows, those strategies cannot but exceeds one cycle in the traditional IFU architecture. On the contrary, in ABSA, if the time to implement the tasks of each stage exceeds one cycle, the only modification is to increase the depth of the related stage. For example, the analysis stage is split into two stages for fitting the branch predictor with a larger array.

Fetch bandwidth is another important characteristic of IFU. Trace cache, as the most popular approach, is widely used in modern processors in order to improve the fetch bandwidth [21], [22], [25]. ABSA can still maintain power efficiency when using trace cache technique. In the traditional IFU, trace cache may increase the power consumption of IFU because the processor simultaneously accesses to both the trace cache and the I-Cache. Otherwise, to avoid this simultaneous access, the sequential trace cache is employed to achieve lower power consumption, but it suffers from a significant performance loss at the meantime. In ABSA, trace cache can be assigned in fetch stage. In the analysis stage, the index bits from the fetch address identifies whether the current trace exists. If the trace cache hits, it can be accessed in the fetch stage and I-Cache access is avoided. Otherwise, I-Cache is normally accessed. Using trace cache in ABSA cannot increase the power consumption since only one of the caches can be accessed in the fetch stage, and the sequential access for a trace cache and I-Cache is avoided so there is no performance degradation.

Furthermore, ABSA is expected to be a power-aware IFU design, which not only is useful in the power reduction of I-Cache, but also can associate with other low-power techniques to effectively reduce the power consumption caused by other power consumers in IFU (e.g., branch prediction, ITLB). In this paper, we only focus on power-saving on I-Cache while ABSA is employed. In fact, the power-saving approaches to reduce the power consumption due to other power consumers of IFU can be also employed by ABSA. Those low-power strategies need carry out analyze or prediction before accessing the corresponding components of IFU, which often seem to require more information and time. In the traditional IFU, since I-Cache, the branch prediction, and ITLB are simultaneously implemented as soon as the fetch address is generated, the process will have to reduce the frequency or increase the design complexity in order to gain more time for low-power execution. On the contrary, in ABSA, the only work designers have to do is adding a new function block into the analysis stage to provide more useful information, or deepening the analysis stage in the case of more time being required by the low-power implementation.

5.3 Area Overhead

ABSA reassigns the tasks for each stage, namely ABSA changes little the function block of the traditional IFU (e.g., branch predictor, SRAM). Three major area increments are an extended BTB, pipeline registers and a voltage control circuit for each I-Cache line. The 1-bit Change Page Field and 5-bit Sentry Target for each entry of BTB increase

0.75 KB area for the 1 K entries BTB. About the pipeline registers, compared to the long bits of instructions and large cache size, it is much smaller. At last, when using the DVS technique, the total area overhead is less than 3% for the entire cache line [13]. Therefore, the area increased by ABSA can fairly be negligible for most superscalar processors.

Furthermore, ABSA employs the subbanking technique, resulting in the increase in the cache size. The area overhead of subbanks mainly includes three parts: 1) Each subarray requires a separate row decoder to reduce the wordline drive and delay in our paper. Because of using the dynamic NOR decoder for row decoders, the area overhead of row decoders is 4144 transistors. 2) ABSA requires a separate modified precharge circuit shown in Fig. 4 for each 1 K subarray. Each precharge circuit requires three PMOS. The total number of the transistors of precharge circuit is 96. 3) In the case of ABSA_bal, two subarrays may be awaken at the same time in order to reduce the branch misprediction penalty. Therefore, each subarray has a separate sense amplifier that is active only when the corresponding wakeup single is valid. The size of the sense amplifier is 270 bits, including block size (32 B) + upper tag (14 bits). We use the conventional 6 T memory cell so the total number of the transistors of the sense amplifier is 1620. The sense amplifiers increased from 4 (one per way) to 32 (one per subarray), the increment of which is 45360 transistors. Taking these aspects, the total area overhead due to subbanks is 49600 transistors. For a 32 KB 4-way cache with a block size of 32 B, the cache area spent in the tag and data arrays is approximately $(1024\times19\times6)+(1024\times256\times6) = 1689600$ transistors. Thus, the overhead due to subbanking technique is around 2.9% of the cache area, it is negligible.

6. Experimental Results

6.1 Simulation Methodology

To evaluate the power consumption and performance in he 0.13 μm CMOS technology, we employ a modified version of SimpleScalar [9], incorporating the Wattch framework [9] to model the dynamic power consumption, and the HotLeakage model [30] for the static component. The Wattch simulator built on the SimpleScalar simulation tool set integrates the CACTI [14] timing, power and area models. The main simulation parameters, listed in Table 1, roughly correspond to those in UltraSPARC-III microprocessor [19]. We use a number of integer benchmarks from the SPEC2000 suites benchmarks. All benchmarks were compiled with highest optimization level by the Alpha compiler [26], and were fast-forwarded pass the first 500 millions instructions to bypass initialization and startup code before measured simulation begins. Then, full-detail simulation is performed for next one billion instructions. We used the *ref* input data set. Table 2 shows that for each benchmark, the input set, the percentage of the dynamic conditional branches and the branch prediction accuracy.

We also model a baseline processor. The IFU architecture of the baseline processor is based on the UltraSPARC-III microprocessor with a 32 KB 4-way I-Cache. To compare the power efficiency with ABSA, two low-power techniques on I-Cache (*i.e.*, two-level filter scheme and drowsy I-Cache) are used in the baseline processor. Besides, the baseline processor does not employ other power-saving approaches in IFU for fair comparison. Note that the branch predictor uses a Gshare algorithm that maybe not advance branch prediction scheme. However, if the simulated processor using such predictor does not suffer from much runtime overhead, others using more accurate branch predictor would suffer from less runtime increment.

Table 1 Simulation processor configuration.

parameters	value
Fetch/Decode/Issue/Commit	4 Instructions Width
Branch Direction Predictor	16 K-entry Gshare
Branch Target Buffer	512-Entry, 2-Way
LSQ Size	32
Instruction Fetch Queue Size	32
Functional Units	4 Int ALU, 2 Int mult/div, FP ALU, 2 FP mult/div, 2 MEMPORT
Branch Misprediction Penalty	6 cycles
ITLB	32entry in each way, 4 KB page size, 4way, LRU, Latency: 30 cycle
Inst./Data L1 Caches	32 KB, 32 Byte Blocks, 4-way Mapped, Latency: 1 cycle
UL2 Cache	256 MB(s), 64 Byte Blocks, 8-way Mapped, Latency: 6 cycle
Memory	Ideal size, Latency: 100 cycle
confidence estimator	4 K-entry, 4 bit JRS (Jacobsen et al. 1996) [15]

Table 2 Benchmark characteristics.

Bench.	Input Set	dynamic conditional branch (%)	Prediction Accuracy (%)
bzip2	ref.graphic	4.14	92.21
crafty	crafty.in	7.66	92.11
eon	kajiya image	4.31	92.64
gap	ref	8.79	94.32
gcc	scilab.in	10.02	93.51
gzip	ref.graphic	7.76	91.26
mcf	ref	8.52	92.10
parser	ref	7.76	91.90
perlbmk	ref.perfext	9.40	91.25
twolf	ref	4.28	86.27
vortex	ref	9.68	97.66
vpr	route.in	7.31	89.96

6.2 Power Savings

In this section, we explain how much power can be saved in ABSA by comparing dynamic and static power consumption of I-Cache with the baseline processor integrating two-level filter scheme and drowsy I-Cache, respectively. The

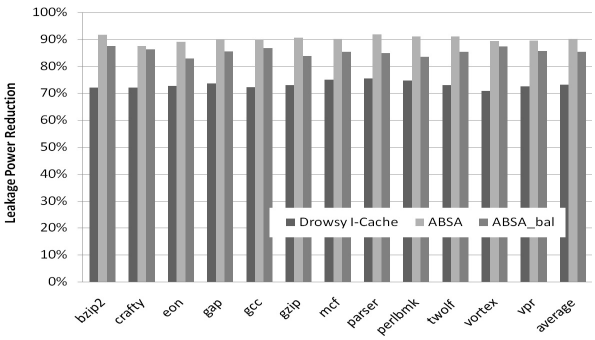

Fig. 7 Static power saving.

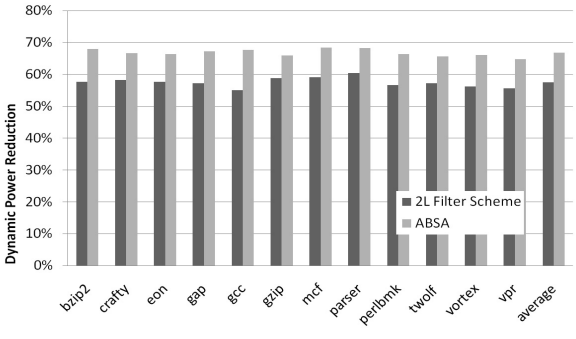

Fig. 8 Dynamic power saving.

main reason to choose these two low-power techniques is because they achieve better power/performance tradeoffs than most other related approaches and both of them are hardware-only approaches, which do not need software supports and change of instruction set architecture.

6.2.1 Static Power Reduction of I-Cache

As shown in Fig. 7, ABSA reduces leakage power in I-Cache data cell by 90.12%, on average. This is because ABSA is a very fine-grained method to reduce I-Cache leakage power. ABSA_bal described in Sect. 5.1 reduces the leakage power by 85.63%, on average. The 4.49% increment in leakage power is because that the wakeup stage needs to awake an extra subarray containing the instructions from the alternative paths following a low-confidence branch. Figure 7 also shows the Drowsy I-Cache with the horizontal configuration, which uses DVS and sub-bank prediction technique, reduces the leakage power by 73.09%, on average. As a result, ABSA is much more effective in saving the leakage power than the Drowsy I-Cache.

6.2.2 Dynamic Power Reduction of I-Cache

I-Cache employed by ABSA models a technique similar to cache subbanking for saving power in the data and tag arrays. Besides reducing power consumption, smaller subarrays also enable the cache array to be as well as possible to minimize wire capacitance, which results in faster access time and lower power dissipation. However, since each subarray has its own row decoder for the low wordline drive and delay, more subarrays mean more power consumption by row decoders. Our measurements show the power consumption by the row decoder is about 22% of the total power consumption of I-Cache in baseline cache configuration. But, in ABSA, only one required row decoder of the selected cache way is activated so the data and tag array can be further partitioned into smaller subarrays. Figure 8 shows the dynamic power saving is about 66.92% for ABSA and 57.62% for two-level filter scheme, respectively. So, even though the two level filter scheme is very effective in reducing I-Cache dynamic power, ABSA can reduce more power than it. It is because that the two-level filter scheme accesses the row decoders in all cache ways during implementing the L1 and L2 filter.

Table 3 Breakdown of power consumption (mWatts).

	Baseline		ABSA	
Branch predictor	235	22.25%	245	33.24%
I-cache	646	61.17%	208	28.22%
Other	175	16.57%	175	23.74%
Low-confidence	0	0.00%	91	12.35%
Pipeline registers	0	0.00%	18	2.44%
Total	1056		737	

6.2.3 Power Saving of IFU

Besides the power consumption of I-Cache, the total power consumption of ABSA includes of other function blocks. Compared to the baseline IFU architecture, ABSA adds three newly hardware that lead to power increment: 1) 6-bits for each BTB entry are required to save sentry tag field and change page field, which increases the branch prediction about 4.3% power consumption, on average. 2) The additional registers are required between the pipeline stages. For wakeup signals, ABSA needs 32 registers (4 ways $\times$ 8 subbank) to transfer the wakeup signal from the analysis stage to the wakeup stage, and another 32 registers from the wakeup stage to the fetch stage. However, the power consumption of those registers is very small because most of them are inactive for a long period until the corresponding subarray needs to be accessed. 3) ABSA_bal also needs a low-confidence mechanism, which causes extra power consumption. In all, compared to the baseline IFU, the added hardware increases the power consumption by 2.65% for ABSA and 11.17% for ABSA_bal.

The breakdown of average power consumption of ABSA and the baseline IFU is shown in Table 3. The simulation results shows, although the newly added hardware consumes some power, the total IFU power is still reduced by 30.3%, on average. Another observation is that I-Cache is no longer the largest contributor of power consumption in ABSA. Note that the branch predictor and other function blocks like as ITLB have not employed any power-saving approaches in our experimentation. Thus, the total power consumption of a processor using ABSA will be further decreased by integrating more low-power techniques.

Table 4 I-Cache access time.

	Baseline (ns)		ABSA (ns)	
	Data path	Tag path	Data path	Tag path
Row decoder	0.35	0.16	0	0
Wordline bitline	0.13	0.06	0.13	0.05
Sense amplifier	0.07	0.05	0.07	0.04
Tag compare	0	0.16	0	0.14
Mux driver	0	0.13	0	0.13
Output driver	0.06	0.06	0.06	0.06
Total	0.61	0.62	0.26	0.42
Total access time	0.62		0.42	

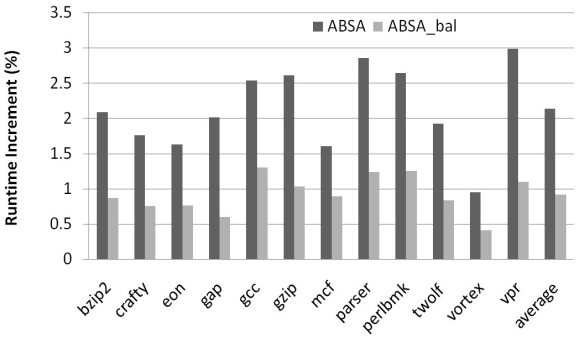

Fig. 9 Runtime increment for each benchmark.

6.3 Performance Analysis

In this section, we analyze the critical path of ABSA, and make quantitative analysis on the performance degradation due to ABSA.

6.3.1 Critical Path

In the analysis stage, the time to generate a wakeup signal consists of the subbank selection time, the way selection time and the corresponding logic control time. By using the CACTI tool, we estimate the time of subbank selection is approximately 0.08 ns. The time of way selection is approximately 0.45 ns (0.35 ns for accessing sentry tag array and 0.1 ns for sentry tag comparison), and the time of control logic is approximately 0.1 ns. Since the way selection signal and the subbank selection signal occur at the same time, the total time to generate a wakeup signal is 0.55 ns. Meanwhile, the time to implement branch prediction and block buffer is 0.46 ns and 0.3 ns, respectively. Thus, the critical path of the analysis stage is 0.55 ns.

In the wakeup stage, according to the report in [13], the transition time switching between the normal mode and drowsy mode is 0.28 ns. Simultaneously, the time of the row decoder is 0.26 ns. In the fetch stage, the time to access I-Cache is shown in Table 4, in detail. The baseline processor spends 0.63 ns for an I-Cache access. ABSA removes the row decode and the part of tag comparison from the fetch stage, so accessing I-Cache in ABSA is only 0.43 ns.

After all, the largest critical path in ABSA is 0.55 ns at the analysis stage. Compared to the baseline IFU whose critical path is 0.62 ns at the I-Cache access, the ideal improvement of processor frequency can achieve 11.29%. Note that the largest delay does not come from I-Cache access, but analysis stage. Therefore, smaller critical path can be expected by further deepening the analysis stage.

6.3.2 Overall Performance

ABSA induces some runtime increment due to larger branch misprediction penalty. In Sect. 5.1, we discussed two approaches to alleviate this penalty. Specifically, ABSA_bal that pre-awakes the subarray for the alternative path can reduce the extra branch misprediction penalty due to ABSA. But, the performance is still reduced because of misprediction of branch prediction and low-confidence mechanism. Figure 9 illustrates the runtime increment for each benchmark. The results show that the runtime in ABSA increases inversely with the branch prediction accuracy and the number of the dynamic branches in each program. For example, the vortex's runtime increment is small because of its high branch prediction accuracy. Although the twolf's branch prediction accuracy is the lowest in all the benchmark, its runtime increment is not the largest. It is because that the number of dynamically executed branches in twolf is small so it encounters the small branch misprediction penalty. ABSA_bal can reduce the branch misprediction penalty due to ABSA, the penalty reduction for each benchmark is based on its own accuracy of the low-confidence mechanism. The highest accuracy is 67.64% for vortex, the lowest is 41.21% for eon and the average accuracy is about 56.27%. Compared to the baseline, ABSA and ABSA_bal increase the simulation cycles by an average of 2.14% and 0.97%, respectively.

7. Conclusions

In this paper, ABSA, a power-efficient IFU architecture, is proposed to reduce the power consumption of I-Cache with little performance overhead. It restructures the IFU pipeline stages and carefully assigns tasks for each stage, which provides sufficient time and information to implement a more accuracy and efficient I-Cache access. The contribution of this paper is that, by removing the unnecessary restrictions on the traditional IFU, ABSA provides the low-power strategies with sufficient time and information to maximize the power efficiency of IFU with a little change in these low-power strategies and hardware modification.

Furthermore, our proposed ABSA ensures compatibility with other low-power approaches and enables them to more effectively reduce the power consumption without the cost of frequency or design complexity. In all, compared to a conventional IFU design, ABSA reduces about 30.3% power consumption of IFU (including 66.92% dynamic I-Cache power reduction and 85.63% static I-Cache power reduction respectively). Meanwhile the performance degradation is about 0.97% while ABSA_bal is employed in the

case of the branch prediction and low-confidence mechanism with low prediction accuracy.

Acknowledgments

This research was supported by CREST, JST and partially by a grant of Regional Innovation Cluster Program 2nd stage, MEXT.

References

[1] J.F. Edmondson, et al., "Internal organization of the alpha 21164, a 300-MHz 64-bit quad-issue CMOS RISC microprocessor," Digital Technical Journal, vol.7, no.1, 1995.

[2] J. Montanaro and et al., "A 160-mhz, 32-b, 0.5-w cmos risc microprocessor," Digital Technical Journal, Digital Equipment Corporation, vol.9, 1997.

[3] G. Hinton et al., "The microarchitecture of the pentium 4 processor," Intel Technical Journal, vol.Q1, Feb. 2001.

[4] J. Kin, M. Gupta, and W.H. Mangione-Smith, "The filter cache: An energy efficient memory structure," Proc. 30th Int. Microarchitecture Symp., pp.184–193, Dec. 1997.

[5] N. Bellas, I.N. Hajj, C.D. Polychronopoulos, and G. Stamoulis, "Architectural and compiler techniques for energy reduction in high-performance microprocessors," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.8, pp.317–326, June 2000.

[6] C.L. Su and A.M. Despain, "Cache design for energy efficiency," Proc. 28th Int. System Sciences Conf., pp.306–315, 1995.

[7] K. Ghose and M.B. Kamble, "Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation," Proc. Int. Low Power Electronics and Design Symp., pp.70–75, 1999.

[8] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," Proc. 27th Annual Intl. Symp. on Computer Architecture, pp.83–94, 2000.

[9] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," IEEE Comput., vol.35, no.2, pp.59–67, Feb. 2002.

[10] M.D. Powell, A. Agarwal, T.N. Vijaykurnar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," Proc. Int. Symposium on Microarchitecture, pp.54–65, Austin, Texas, USA, 2001.

[11] Y. Cllarlg, S. Ruan, and F. Lai, "Design and analysis of low-power cache using two-level filter scheme," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.11, no.4, pp.568–580, 2003.

[12] K. Flautner, et al., "Drowsy caches: Simple techniques for reducing leakage power," ACM SIGARCH Computer Architecture News, vol.30, May 2002.

[13] N.S. Kim, K. Flautner, D. Blaauw, and T. Mudge, "Energy efficient memory systems: Drowsy instruction caches: Leakage power reduction using dynamic voltage scaling and cache sub-bank prediction," Proc. 35th annual International Symposium on Microarchitecture, Nov. 2002.

[14] P. Shivakumar and N.P. Jouppi, "CACTI 3.0: An integrated cache timing, power, and area model," http://www.hpl.hp.com.techerports/PCompaq2DEC/WRL-2001-2.html, 2001.

[15] E. Jacobsen, E. Rotenberg, and J.E. Smith, "Assigning confidence to conditional branch predictions," Proc. 29th Annual Symp. And Workshop on Microprogramming and Microarchitecture (MICRO-29), 1996.

[16] J.E. Smith and G.S. Sohi, "The microarchitecture of superscalar processor," Proc. IEEE, vol.83, no.12, pp.1609–1624, 1995.

[17] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," Proc. 28th annual International Symposium on Computer Architecture, May 2001.

[18] M. Powell, et al., "Gated-vdd: A Circuit technique to reduce leakage in deep-submicron cache memories," Proc. International Symposium of Low power Electronics and Design, 2000.

[19] T. Horcl and G. Lautcrbach, "UltraSPARCIII: Designing third generation 64-bit performance," IEEE Micro, vol.19, no.3, pp.73–85, 1999.

[20] H.Q. Le, W.J. Starke, J.S. Fields, F.P. O'Connell, D.Q. Nguyen, B.J. Ronchetti, W.M. Sauer, E.M. Schwarz, and M.T. Vaden, "IBM POWER6 microarchitecture," IBM Journal of Research and Development, vol.51, Issue 6, pp.639–662, 2007.

[21] G. Hinton, D. Sager, M. Upton, D. BVoggs, D. Carmean, A. Kyker, and P. Roussel, "The microarchitecture of the pentium4 processor," Intel Technology Journal, Q1 2001, 2001.

[22] P. Bannon and Y. Saito, "The Alpha 21164 microprocessor," Proc. COMPCON, pp.389–398, San Francisco, 1997.

[23] S. McGeady, "The i960CA superscalar implementation of the 80960 architecture," Proc. COMPCON, pp.232–240, San Francisco, 1990.

[24] E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," Proc. 29th Annual International Symposium on Computer Architecture, pp.25–34, May 2002.

[25] E. Rotenberg, S. Bennett, and J.E. Smith, "Trace cache: A low latency approach to high bandwidth instruction fetching," Proc. 29th International Symposium on Microarchitecture, pp.24–34, Dec. 1996.

[26] E. Perelman, G. Hamerly, and B. Calder. "Picking statistically valid and early simulation points," Proc. 2003 Intl. Conf. on Parallel Architectures and Compilation Techniques, pp.244–255, Sept. 2004.

[27] R. Min, Z. Xu, Y. Hu, and W.-B. Jone, "Partial tag comparison: A new technology for power-efficient set-associative cache designs," VLSID04: 17th International Conference on VLSI Design, pp.183–188, Jan. 2004.

[28] S. Wilton and N. Jouppi, "An enhanced cache access and cycle time model," IEEE J. Solid-State Circuits, vol.31, no.5, pp.677–688, 1996.

[29] J.P. Shen, Modern Processor Design: Fundamentals of Superscalar Processors, chapter 3.5, McGraw-Hill Science/Engineering/Math, 1 edition, July, 2004.

[30] Y. Zhang, D. Parikh, K. Sankaranarayanan, et al., "HotLeakage: a temperature-aware model of subthreshold and gate leakage for architects," University of Virginia, Department of Computer Science, Technical Report CS-2003-05 Report, March 2003.

Jiongyao Ye was born in ShangHai, China on May, 1978. He received the B.E degree in Electronic Engineering in 2000, from Shanghai Marine University, where he was an assistant during 2000-2002. In 2005, he received the M.S. degree in Graduate School of Information, Productions and Systems, from Waseda University. He then joined the Sony LSI Design Inc., where he worked in the field of LSI design from 2005 to 2008. He is currently working toward the Dr. Eng. degree in Graduate School of Information, Productions and Systems, from Waseda University. His research interests include micro-architecture, low-power FPGA and their applications.

Yingtao Hu was born in Wuxi, China on May, 1986. He received the B.E. degree in software engineering from Dalian University of technology, China in 2008. He is currently a graduate student in the Graduate School of Information, Production and Systems, Waseda University, Japan. His research interest includes low-power LSI design and information security.

Hongfeng Ding was born in Liaoyan, China on April, 1986. He received the B.E. degree in mathematics and applied mathematics from Beijing University of Posts and Telecommunications, China in 2009. He is currently a graduate student in the Graduate School of Information, Production and Systems, Waseda University, Japan. His research interest includes high performance processor design and low-power design.

Takahiro Watanabe was born in Ube, Japan on October, 1950. He received the B.E. and the M.E. in Electrical Engineering from Yamaguchi University, and the Dr. Eng. from Tohoku University. In 1979, he joined Research and Development Center of TOSHIBA Corp., where he worked in the field of LSI design automation. In August 1990, he joined Yamaguchi University, the Department of Computer Science and Systems Engineering, and in April 2003, he moved to Waseda University, Graduate School of Information, Production and Systems. His current research interests are EDA algorithm, Microprocessor and MPSoC, NoC, FPGA and their applications. He is a member of IPSJ and IEEE.