CASA: A New IFU Architecture for Power-Efficient Instruction Cache and TLB Designs

Han-Xin Sun, Kun-Peng Yang, Yu-Lai Zhao, Dong Tong, and Xu Cheng

Microprocessor Research and Development Center, Peking University, Beijing 100871, China

E-mail: {sunhanxin,yangkunpeng,zhaoyulai,tongdong,chengxu}@mprc.pku.edu.cn

Received January 7, 2007; revised August 9, 2007.

Abstract The instruction fetch unit (IFU) usually dissipates a considerable portion of total chip power. In traditional IFU architectures, as soon as the fetch address is generated, it needs to be sent to the instruction cache and TLB arrays for instruction fetch. Since limited work can be done by the power-saving logic after the fetch address generation and before the instruction fetch, previous power-saving approaches usually suffer from the unnecessary restrictions from traditional IFU architectures. In this paper, we present CASA, a new power-aware IFU architecture, which effectively reduces the unnecessary restrictions on the power-saving approaches and provides sufficient time and information for the power-saving logic of both instruction cache and TLB. By analyzing, recording, and utilizing the key information of the dynamic instruction flow early in the front-end pipeline, CASA brings the opportunity to maximize the power efficiency and minimize the performance overhead. Compared to the baseline configuration, the leakage and dynamic power of instruction cache is reduced by 89.7% and 64.1% respectively, and the dynamic power of instruction TLB is reduced by 90.2%. Meanwhile the performance degradation in the worst case is only 0.63%. Compared to previous state-of-the-art power-saving approaches, the CASA-based approach saves IFU power more effectively, incurs less performance overhead and achieves better scalability. It is promising that CASA can stimulate further work on architectural solutions to power-efficient IFU designs.

Keywords computer architecture, instruction cache, instruction TLB, instruction fetch unit, power-efficient design, dynamic voltage scaling

1 Introduction

The instruction fetch unit (IFU) is one of the most attractive targets for power-efficient designs. The large arrays of the branch predictor, instruction cache (I-Cache) and instruction TLB (ITLB) usually dissipate a considerable portion of power in modern processors. In commercial processors such as Alpha 21264[1], Pentium Pro[2] and Strong-ARM SA110[3], IFU alone consumes 23%, 14% and 36% of total chip power, respectively. More severely, with the continuous technology scaling, leakage power will soon dominate the total power dissipation in the processor. In particular, leakage will account for 70% of I-Cache power budget in 70nm technology if left unchecked[4], making it more complex to control the IFU power consumption. If no effective power-saving approach is employed, IFU will probably achieve high power density and become a "hot spot", which makes it much more difficult to guarantee the chip's reliability without sacrificing performance.

The goal of this paper is to effectively reduce I-Cache and ITLB power with minimal performance overhead. Our work is based on the observations on traditional IFU architectures and previous power-saving approaches. Recently, many approaches have been proposed to reduce the power dissipations in I-Cache and ITLB. Before accessing I-Cache or ITLB, they usually need to identify and eliminate some unnecessary accesses. However, in traditional IFU architectures, since I-Cache and ITLB are accessed in the earliest pipeline stage, the fetch address needs to be sent to I-Cache and ITLB arrays for instruction fetch as soon as it is generated. Therefore, limited work can be done by the power-saving logic after the fetch address generation and before the instruction fetch. Due to this restriction from traditional IFU architectures, previous power-saving approaches often lose the opportunities to maximize power efficiency, and induce extra delay or increased runtime. This problem, which
in turn challenges traditional IFU architectures, stimulates us to seek a better power-aware IFU architecture.

In this paper, we propose CASA, a new IFU architecture, which performs Careful Access Selections Ahead of ITLB and I-Cache lookups. Early in the front-end pipeline, the fetch address is carefully analyzed to determine the necessity of address translation before ITLB lookup and to select the set and way before I-Cache access. In addition, a separate pipeline stage is added to only turn up the supply voltage of the I-Cache line required by fetch address and keep others in low-voltage mode. As an integrated method to reduce the power of both I-Cache and ITLB, CASA provides sufficient time and information for the power-saving approaches to identify and eliminate most unnecessary accesses, and it does not suffer much from problems such as timing overhead, runtime increase or poor scalability. Specifically, the processor benefits from CASA in the following four major aspects.

(a) CASA reduces I-Cache leakage power by 89.7%, which shows almost the perfect effectiveness in I-Cache leakage power reduction.

(b) CASA saves I-Cache dynamic power by 64.1% and ITLB dynamic power by 90.2%, which is more effective than two-level filter scheme[5] and comparable to the Hardware-Only Approach in [6].

(c) The performance degradation in the worst case is only 0.63%, which is negligible.

(d) CASA is more scalable than traditional IFU architectures for higher fetch bandwidth and frequency.

The rest of this paper is organized as follows. We discuss previous power-saving approaches and analyze traditional IFU architectures in Section 2. We present the CASA architecture in Section 3. We describe the power-efficient design of CASA-based IFU in Section 4. We discuss the performance and scalability of CASA-based IFU in Section 5. We show the experimental results in Section 6. In Section 7, we conclude this paper and discuss the future work.

2 Related Work

There is a large body of work that reduces the dynamic power dissipation in the cache. L-cache[7], block buffer[8] and multiple line buffers[9] employ a small cache between the processor and level one cache to avoid unnecessary level one cache lookups. Cache sub-banking techniques[8,9] divide the data memory arrays into multiple sub-banks, and access only the sub-bank which contains the data desired by the address. Way-prediction[10], a more popular approach, 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[5], which accesses a block buffer and Sentry-Tag arrays ahead of cache. First, if the block buffer results in a hit, the accesses to cache arrays are avoided. Moreover, if a Sentry-Tag array causes a miss, the access to the data array of corresponding cache way is avoided. By report, it reduces the accesses to the 32KB two-way set-associative I-Cache by 81.13%. Although this approach is very effective in saving power, the two-level filter induces some timing overhead. In this paper, the idea of the two-level filter scheme is employed to analyze the requirement of the fetch addresses, but it induces minimal timing overhead in the CASA IFU architecture.

As feature size shrinks, reducing cache leakage power becomes an important design goal. State-preserving techniques, such as Gated-\(V_{DD}\)[11], do not preserve circuit states in sleep mode. Since data is lost in sleep mode, they suffer from the increased runtime to re-fetch data from lower level of memory. On the contrary, the state-preserving techniques proposed in [4, 12~16] preserve circuit states in sleep mode. Drowsy I-Cache[4], a representative state-preserving technique, employs dynamic voltage scaling (DVS) circuit technique and sub-bank prediction scheme to speculatively wake up one I-Cache sub-bank and keep others in the low-voltage mode. It reduces leakage power consumption in I-Cache by about 75%. However, since it highly depends on the accuracy of the sub-bank predictor, it induces both performance degradation and indefinite cache hit time. For some benchmarks, the runtime increases are considerable.

Researchers have also paid attention to the power reduction of ITLB. Lee et al.[17] have proposed a filter-bank TLB system to reduce the number of entries accessed at a time. Several approaches that reuse the physical address generated previously to avoid ITLB lookups have also been proposed[6,18]. Since these approaches use a comparator to detect the change of virtual page number before accessing ITLB, they suffer from the timing overhead of the comparator.

Although these approaches can reduce the power consumptions of I-Cache and ITLB, we find that they are still restricted by traditional IFU architectures. Before accessing I-Cache or ITLB, they usually require some time and information to identify and eliminate some unnecessary accesses. However, traditional IFU architectures do not quite satisfy this requirement. Historically, the IFU architecture accesses the branch predictor after I-Cache and ITLB[19], as shown in Fig.1(a). The alternative IFU architecture, in processors such as UltraSPARC-III[20], accesses branch
predictor in parallel with I-Cache and ITLB to reduce the branch delay, as illustrated in Fig.1(b). In both of the traditional IFU architectures, since I-Cache and ITLB are accessed in the earliest pipeline stage, the fetch address needs to be sent to I-Cache and ITLB arrays as soon as it is generated. Therefore, quite limited spare time and information are provided for the power-saving approaches to identify the unnecessary accesses after the fetch address generation and before the instruction fetch.

Due to the restrictions from traditional IFU architectures, the power-saving approaches are prevented from achieving a better power/performance tradeoff. First, since limited spare time and information are provided, they often lose the opportunities to maximize power efficiency. Moreover, when attempting to use more time and information to identify and eliminate unnecessary accesses, these approaches often induce extra delay or runtime increase. These problems, which reflect that traditional IFU architectures are not quite suitable for applying power-saving approaches in I-Cache and ITLB designs, gradually challenge the traditional IFU architectures as power efficiency becomes increasingly important for IFU designs. To achieve a better power/performance tradeoff, a new IFU architecture which is more suitable for applying power-saving approaches is in demand.

3 CASA Architecture

From the observations in Section 2, we are motivated to propose a new IFU architecture, which considers the components in the IFU as a whole and removes unnecessary restrictions on the power-saving approaches. Our design methodology consists of the following three major principles.

(a) Using a separate pipeline stage to generate the information for power-saving logic. If we use a separate pipeline stage for the analysis of the fetch address before accessing I-Cache and ITLB, sufficient time and information can be used by the control logic to perform a Careful Access Selections Ahead of I-Cache and ITLB lookups, which will provide the opportunity to maximize the power efficiency.

(b) Providing a separate pipeline stage for waking up I-Cache. This principle is based on the dynamic voltage scaling (DVS) circuit technique, which is popularly used for I-Cache leakage power reduction. When I-Cache will be accessed, this technique wakes up part of I-Cache by changing the supply voltage to normal $V_{DD}$. When I-Cache need not be accessed, it changes the supply voltage to a lower $V_{DD}$, reducing the leakage power consumption. We not only employ this technique but also provide a separate pipeline stage for it to change the memory cells from low-voltage mode to normal-voltage mode.

(c) Using the provided stages and information to accurately access I-Cache and ITLB. By careful analysis of the fetch addresses, the control of I-Cache and ITLB lookups can be more accurate than previous approaches, and most unnecessary accesses to I-Cache and ITLB can be avoided.

According to the above design methodology, the proposed IFU architecture, CASA, is made up of three stages: ANALYSIS, WAKEUP and FETCH, as shown in Fig.2.

3.1 ANALYSIS Stage

The key of CASA is the ANALYSIS stage, which uses various building blocks to analyze the fetch address, as illustrated in Fig.3: 1) set decoder of I-Cache;
2) two-level cache filter\cite{5}; 3) extended BTB; 4) ITLB.

The I-Cache set decoder is used to decide which cache set needs to be accessed. Different from traditional designs, the set decoder in CASA is located at the ANALYSIS stage, two stages ahead of I-Cache data arrays.

The two-level cache filter\cite{5} is used to decide which cache way needs to be accessed. First, one block buffer is used for filtering accesses to the whole I-Cache. If the block buffer has contained the instructions to be fetched, the access to I-Cache is avoided. In addition, each I-Cache tag array is divided into two parts. The first part, named Sentry-Tag array, contains five lower bits of the tags. The second part, named Upper-Tag array, contains the remaining tag bits. Note that, different from two-level filter scheme, the block buffer and Sentry-Tag arrays in CASA are located two stages ahead of the Upper-Tag and data arrays, as illustrated in Fig.3. For each cache way, if the Sentry-Tag array identifies a miss according to the comparison result between Sentry-Tag and fetch address in the ANALYSIS stage, the access to the Upper-Tag and data arrays of that cache way in the FETCH stage will be avoided.

It is well understood that BTB can maintain not only the target addresses and instructions but also some other useful information for many branch instructions. To better steer the accesses to I-Cache and ITLB, we extend the BTB in the ANALYSIS stage. The detailed design of the extended BTB will be described in Subsection 4.1.

Similar to traditional designs, the ITLB in CASA translates the virtual page number to physical page number.

Note that based on the CASA architecture, one can employ some other building blocks in the ANALYSIS stage to perform even more detailed analysis of the fetch address, which probably brings higher power efficiency.

3.2 WAKEUP and FETCH Stages

Based on the DVS technique, the WAKEUP stage selectively wakes up the I-Cache lines according to the address analysis results from the ANALYSIS stage. Instead of waking up the I-Cache sub-bank\cite{4}, this stage implements the DVS circuit technique for each cache line, as illustrated in Fig.4. For each cache line, the “Line wakeup” signal is controlled by the analysis result. When a cache line will be accessed, the “Line wakeup” signal is asserted and the cache line is woken up by a normal $V_{DD}$. When the cache line need not be accessed, its supply voltage is switched to a low $V_{DD}$, which reduces its leakage power consumption.

4 CASA-Based Power-Saving Approach

Based on the CASA architecture, we will show how the IFU is designed to effectively save I-Cache and ITLB power in this section. We will first present the methods to extend and update BTB in Subsection 4.1. Then we will describe the way to reduce ITLB and I-Cache power in Subsections 4.2 and 4.3.

For the reduction of I-Cache leakage power, I-Cache dynamic power and ITLB dynamic power, we compare our approach with drowsy I-Cache\cite{4}, two-level filter scheme\cite{5} and Hardware-Only Approach in \cite{6}, respectively. We choose these approaches for comparison because: 1) they are state-of-the-art approaches because they achieve better power/performance trade-offs than most other related approaches; 2) they are all hardware-only approaches, which do not need the change of software.

4.1 Extending BTB Entries

In the ANALYSIS stage, BTB is extended by more information about the branch instructions. As shown in Fig.5, each entry of extended BTB consists of the following fields: 1) Valid: recording whether the entry is valid; 2) Tag: most significant bits of the address of branch instruction; 3) Target: virtual target address of branch instruction; 4) Change-Page: one bit, recording whether the branch instruction will change to another physical page when it is taken (“1” for change, “0” for not change); 5) Sentry-Target: for the branch target instruction, it records the same bits of the physical page number as the I-Cache Sentry-Tag. Thus, when the instruction flow changes to the branch target instruction, Sentry-Target can be directly compared to
I-Cache Sentry-Tag to predetect whether the corresponding cache way will cause a miss. Previously, the method of extending BTB with Way-Pointer to reduce the accesses to I-Cache ways have been used[21]. However, combined with Sentry-Tag arrays, the extension with Sentry-Target is superior because it does not need the Way-Pointer invalidation whenever a cache replacement takes place.

Due to the extension, the BTB update mechanism is slightly different from the traditional one. When a branch is resolved, the BTB is updated only if it has resulted in a miss for that branch. When BTB needs update, ITLB is accessed by the virtual address of branch target instruction (VA<sub>bt</sub>) and translates VA<sub>bt</sub> into the physical address of the branch target instruction (PA<sub>bt</sub>). As illustrated in Fig.6, PA<sub>bt</sub> is used to generate both Sentry-Target and Change-Page fields. The Sentry-Target field is just the corresponding bits of PA<sub>bt</sub>, and the Change-Page field is the comparison result between the page number of the branch’s physical address (PA<sub>b</sub>) and that of PA<sub>bt</sub>.

![Fig.6. Update of extended BTB.](image)

4.2 Reducing ITLB Power

To avoid unnecessary ITLB lookups, the page number generated last cycle is recorded by Last Physical Page Number (LPPN) register. If current virtual page number is detected to be the same as the last one, the access to ITLB is not enabled and LPPN is reused as the current physical page number.

Understanding how we can detect the change of virtual page number requires a closer look at the change of fetch address. For most programs, fetch address (FA) changes mainly in two cases. First, it can be sequentially increased by the fetch width (FASeq). In this case, we employ a comparator to detect whether FA achieved the end of a virtual page at the last cycle, and the comparison result was recorded by the VP-End register illustrated in Fig.7. If VP-End is 1 at the current cycle, we can find that the instruction flow has just gone from the end of a virtual page to the beginning of a new virtual page. Second, current FA can be the target address of the branch fetched last cycle if the branch direction predictor predicts taken and BTB provides the target address (FABpred). In this case, since the Change-Page field of BTB is generated at the same time as the Target field, which is just FA, we employ it to indicate whether the branch instruction fetched last cycle has just changed instruction flow to a new virtual page. Both the cases of FASeq and FABpred can be easily identified by the processor because it executes the programs.

![Fig.7. Filtering accesses to ITLB.](image)

Compared to the Hardware-Only Approach (HoA) in [6], which compares current virtual page number to the last one, CASA cannot check whether there is a change of virtual page number when a branch mis-prediction or an exception occur. However, since these cases are not frequent, the ITLB dynamic power saved by CASA can be comparable to that saved by HoA.

4.3 Reducing I-Cache Power

4.3.1 I-Cache Access Process

To maximize I-Cache power efficiency, the CASA-based IFU attempts to trigger only the cache line desired by the fetch address, and keep other cache lines in the low-voltage mode. The detailed process to access I-Cache line is illustrated in Fig.8. To trigger only one cache line, the CASA-based IFU employs both set and way selections in the ANALYSIS stage to decide which cache set and which way need to be accessed by the fetch address. The selection results are transferred to the WAKEUP and FETCH stages by pipeline registers and control the “Line wakeup” and “Wordline access” signals to selectively trigger the I-Cache lines. They are also transferred to the FETCH stage to se-
Fig. 8. Process to access an I-Cache line.

The set selection is performed by decoding I-Cache index. As illustrated in Fig. 8, in the ANALYSIS stage, as soon as current FA is generated, a part of it is used as the index of I-Cache arrays. The set decoder translates the index into set selection signals, which are in one-hot encoding format and will activate only the selected cache set. Note that there is only one set decoder in the ANALYSIS stage.

For way selection, the two-level filter is accessed in the ANALYSIS stage, as illustrated in Fig. 8. First, the block buffer is accessed to check if it has contained the required instructions. Meanwhile the Sentry-Tag indexed by FA in each way is compared with corresponding bits of the physical fetch address. Only if the block buffer is miss and the comparison result is equal can the corresponding cache way be selected.

However, the difficulty of the way selection is how to generate the corresponding bits of physical fetch address for Sentry-Tag comparison. To address this problem, we also take a closer look at the change of FA. As illustrated in Fig. 8, in the case of FASeq, the corresponding bits of LPPN can be compared with the Sentry-Tag. Or else, in the case of FABpred, the Sentry-Tag field of BTB, which is generated the same time as the Target field, can be selected for Sentry-Tag comparison.

Generally, if the Sentry-Tag comparison result is not equal, the access to Upper-Tag and data arrays of the corresponding cache way is avoided. However, an exception occurs when VP-End register is 1 in the case of FASeq. As mentioned in Subsection 4.2, in this exceptional case, the instruction flow has just gone across two virtual pages. Therefore, neither LPPN nor Sentry-Target is useful for Sentry-Tag comparison, and Sentry-Tag comparison result is not useful for filtering the access to the cache way (implemented by the OR gate in Fig. 8). Fortunately, since the change of virtual page number is not frequent in the case of FASeq, this exceptional case does not occur frequently.

4.3.2 Effectiveness Comparisons

Using sub-bank prediction scheme for leakage power reduction, drowsy I-Cache has to wake up all cache lines in the predicted sub-bank. Moreover, it highly depends on the accuracy of sub-bank predictor. To improve accuracy, the sub-bank predictor has to increase its complexity, which in turn means probably more power consumption. CASA does not suffer from this problem. In CASA, the set selection is just the decoding of index in ANALYSIS stage, and the effectiveness of the way selection using two-level filter can be demonstrated by [5]. Generally, by set and way selections, only one cache line is woken up. Therefore, we
consider CASA as an “on-demand” approach, which tries to wake up only the cache line on the demand of fetch address. Note that compared to long data and tag bits, power of the circuit to control the supply voltage in each cache line is negligible, as demonstrated in drowsy D-Cache design\cite{4}.

Although the two-level filter scheme\cite{5} is very effective in reducing I-Cache dynamic power, CASA can reduce more power than it. In fact, before filtering any accesses, the two-level filter scheme accesses the row decoders in all cache ways. For a 32KB two-way set-associative cache, the row decoders account for 17.5% of power consumption according to CACTI 2.0 model\cite{22}. In CASA, only one set decoder is accessed in ANALYSIS stage, and the power of accessing multiple row decoders is eliminated.

In all, as an “on demand” approach, CASA can be more effective in saving I-Cache leakage and dynamic power than drowsy I-Cache\cite{4} and two-level filter scheme\cite{5}.

**5 Performance, Scalability and Area Considerations**

In this section, we discuss the performance, scalability and area of CASA-based IFU. We consider performance because power-efficient design approaches should not induce much performance degradation. We consider scalability because the applicability of a new IFU architecture is related to its scalability. If an IFU architecture is not applicable when the processor has higher frequency or higher fetch bandwidth, it cannot be popularly used, either. With the continuous technology scaling, area is often a less important design factor for superscalar processors than power, performance and even scalability. However, to estimate CASA’s hardware cost, we also take it into concern.

**5.1 Performance**

Two opposite factors affect the performance of CASA-based IFU. The reduction of delay on the critical paths of I-Cache and ITLB could potentially improve the processor frequency, because I-Cache and ITLB usually reside on the critical paths of the processor. On the other hand, increasing the front-end pipeline length will decrease the fetch speed when branch misprediction takes place. Both issues are covered in this section, and the quantitative results will be given in Section 6.

**5.1.1 Delay on Critical Paths**

For instruction fetch, the two-level filter scheme\cite{5} increases cache hit time because it accesses the block buffer and Sentry-Tag arrays before cache data arrays in one cycle. On the contrary, CASA reduces two kinds of delay for I-Cache access. First, by moving Sentry-Tag arrays to ANALYSIS stage, less tag bits remain at Upper-Tag arrays in the FETCH stage, reducing the time to access I-Cache tag arrays. Second, by moving set-decoding logic to ANALYSIS stage, the time to access the row decoders in the FETCH stage is eliminated.

For address translation, the Hardware-Only Approach\cite{6} induces considerable delay of the virtual page number comparator. CASA does not suffer from this problem because the logic to enable ITLB lookup in CASA is quite simple, as illustrated in Fig.7.

**5.1.2 Reducing Branch Misprediction Penalties**

The two added stages in CASA may result in larger branch misprediction penalties. This runtime impact can be alleviated by two means. First, we can improve the accuracy of the branch predictor, while mainstream processors, such as the work in \cite{23}, always seek this method. Second, before a branch is resolved, the cache line containing the path alternative to the predicted direction is woken up. When the branch is resolved and misprediction is discovered, since the cache line to be fetched has been woken up, the instruction stream can go directly from ANALYSIS stage to FETCH stage, which reduces the branch misprediction penalties of CASA by one cycle.

Compared to drowsy I-Cache\cite{4}, CASA induces less runtime impact. In fact, besides sub-bank misprediction, drowsy I-Cache also incurs larger runtime increase than CASA when branch misprediction happens. Whenever a branch misprediction happens, drowsy I-Cache needs one more cycle to select the sub-bank to be fetched by the correct address. Moreover, if the selected sub-bank is in drowsy mode, another cycle is required to wake up that sub-bank.

**5.2 Scalability**

When the processor runs at a higher frequency and higher fetch bandwidth, CASA is more scalable than traditional IFU architectures.

As the processor frequency grows, it is possible that the time to wakeup sub-bank exceeds one cycle in drowsy I-Cache\cite{4}, which means the sub-bank prediction needs to be made more cycles before I-Cache is accessed. Hence, more control-flow dependent instructions, which are fetched in these cycles, will affect the sub-bank prediction accuracy. On the contrary, in
CASA, if the time to wakeup I-Cache line exceeds one cycle, the only thing designers need to change is the number of pipeline stages for waking up I-Cache.

When the fetch bandwidth is improved by some components such as trace cache\cite{24}, CASA can still maintain IFU’s power efficiency. In traditional IFU architectures, without the information about whether there is a predicted taken branch, trace cache is aggressively accessed in the earliest pipeline stages, consuming much redundant power. In CASA, trace cache can be just integrated into the FETCH stage, as illustrated in Fig.9, and the outcome of the branch predictor in the ANALYSIS stage is used to steer the accesses to trace cache or I-Cache. Therefore, the instructions at the right path are fetched, while the power efficiency is maintained. Hu et al.\cite{25} have proved the effectiveness of using branch prediction outcomes for power-efficient instruction fetch from trace cache and I-Cache.

![Fig.9. Trace cache in CASA for higher fetch bandwidth.](image)

### 5.3 Area

CASA does increase the area of IFU in two major aspects. First, the Change-Page and Sentry-Target fields increase the width of BTB entries. Second, the pipeline registers and voltage control circuit for each I-Cache line also occupy some area. However, compared to the Tag and Target fields of BTB, the Change-Page and Sentry-Target fields are much shorter. Compared to the long bits of instructions, the pipeline registers and voltage control circuit for each I-Cache line are also much smaller. Therefore, we believe the area increased by CASA can fairly be negligible for most superscalar processors.

### 6 Evaluation

#### 6.1 Experimentation Methodology

To evaluate the power and performance in 70nm CMOS technology, we employ the HotLeakage 1.0\cite{26} power/performance simulator, which is built upon Wattch 1.02\cite{27} power/performance simulator, and has circuit-level accuracy for modeling the leakage current of cache-like structures. The Wattch 1.02 power/performance simulator is built on the SimpleScalar 3.0\cite{28} simulation tool set, and integrates the CACTI 3.0\cite{29} timing, power and area models. The SPEC CPUint 2000\cite{30} benchmarks compiled for Alpha instruction set are run on the simulator. Each benchmark is run for 300 million instructions.

<table>
<thead>
<tr>
<th>Table 1. Simulated Processor Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
</tr>
<tr>
<td>Fetch/Decode/Issue/Commit</td>
</tr>
<tr>
<td>Width</td>
</tr>
<tr>
<td>Branch Direction Predictor</td>
</tr>
<tr>
<td>Branch Target Buffer</td>
</tr>
<tr>
<td>RAS Size</td>
</tr>
<tr>
<td>Instruction Fetch Queue Size</td>
</tr>
<tr>
<td>RUU Size</td>
</tr>
<tr>
<td>LSQ Size</td>
</tr>
<tr>
<td>Functional Units</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Branch Misprediction Penalty</td>
</tr>
<tr>
<td>ITLB/DTLB</td>
</tr>
<tr>
<td>L1 Caches</td>
</tr>
<tr>
<td>L2 Unified Cache</td>
</tr>
<tr>
<td>Memory Latency</td>
</tr>
</tbody>
</table>

The baseline configuration we use is detailed in Table 1. For comparison, we suppose the baseline processor uses the IFU architecture illustrated in Fig.1(b), and does not employ any power-saving approaches in the IFU. To implement the model of CASA-based IFU, the function `ruu_fetch` in file `sim-outorder.c` is changed to CASA architecture. In addition, each BTB entry is extended with more contents, and the way to update BTB is slightly modified to support the extension.

As the performance impact of CASA is dependent on the branch prediction accuracy, we employ a branch direction predictor like that in UltraSPARC-III\cite{20}, which is not too complex. We believe that if the superscalar processor using such predictor does not suffer from much runtime increase, most superscalar processors which usually use more complex predictors will not suffer from much runtime increase, either. The branch prediction accuracy is listed in Table 2.

#### 6.2 Power Savings

In this subsection, we first investigate the I-Cache leakage power reduction. Then we analyze the dynamic power reductions of I-Cache and ITLB. Finally,
Table 2. Branch Prediction Accuracy

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Accuracy</th>
<th>Benchmark</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>bzip2</td>
<td>92.2%</td>
<td>crafty</td>
<td>92.4%</td>
</tr>
<tr>
<td>con</td>
<td>93.1%</td>
<td>gap</td>
<td>94.5%</td>
</tr>
<tr>
<td>gcc</td>
<td>92.6%</td>
<td>gzip</td>
<td>91.7%</td>
</tr>
<tr>
<td>mcf</td>
<td>92.1%</td>
<td>parser</td>
<td>92.8%</td>
</tr>
<tr>
<td>perlbench</td>
<td>91.9%</td>
<td>twolf</td>
<td>87.8%</td>
</tr>
<tr>
<td>vortex</td>
<td>97.7%</td>
<td>vpr</td>
<td>91.4%</td>
</tr>
</tbody>
</table>

we study the power reductions of IFU and processor.

6.2.1 I-Cache Leakage Power Reduction

As illustrated in Fig. 10, on average, CASA reduces the I-Cache cell leakage power by 93.6%. This is because CASA is a very fine-grained method to reduce I-Cache leakage power, as discussed in Subsection 4.3. By comparison, simulation results show that NSPCT, which is a sub-bank prediction method in drowsy I-Cache[4], reduces I-Cache leakage power by 75.4%. Therefore, CASA is much more effective in saving I-Cache cell leakage power than NSPCT.

In addition, we find that the remaining 6.4% of I-Cache leakage power in CASA is consumed mainly in three scenarios: 1) sometimes the two-level filter cannot accurately filter all the unnecessary accesses to cache ways; 2) the cache lines containing alternative path to branch prediction has been woken up, as mentioned in Subsection 5.1.2, but the branch prediction is correct; 3) accessing the cache lines desired by the fetch addresses, which is the essential function of I-Cache. Simulation results show that, in the remaining 6.4% of I-Cache leakage power, approximately 65% is dissipated in Scenario 3 on average, and only 9% and 26% are wasted in Scenarios 1 and 2, respectively. Therefore, CASA is almost perfect in reducing I-Cache cell leakage power.

The bitline leakage power is reduced by 70.1% on average, which is also quite effective. In all, due to both cell and bitline leakage reductions, the leakage power in the I-Cache is decreased by 89.7%. Therefore, CASA reduces most of the I-Cache leakage power.

6.2.2 I-Cache Dynamic Power Reduction

As illustrated in Fig. 11, on average, CASA reduces I-Cache dynamic power by 64.1%. By comparison, simulation results show that the two-level filter scheme[5] reduces I-Cache dynamic power by 58.2%. CASA reduces more I-Cache dynamic power than two-level filter scheme because CASA accesses only one I-Cache set decoder, as discussed in Subsection 4.3.

6.2.3 ITLB Dynamic Power Reduction

As illustrated in Fig. 12, on average, CASA reduces ITLB dynamic power by 90.2%. By comparison, the Hardware-only Approach (HoA)[6] reduces ITLB dynamic power by 95.6%. Compared to HoA, 5.4% (= 95.6% − 90.2%) power in CASA is wasted because CASA cannot check whether the virtual page number of the fetch address will change when the branch mis-prediction or exception occur, as mentioned in Subsection 4.2. Although CASA reduces less ITLB power than HoA, it is also very effective.

6.2.4 Power Reductions of IFU and Processor

To study the power consumption of entire IFU, we
consider not only I-Cache and ITLB power reductions but also the power consumption of added hardware. The major hardware cost of CASA-based IFU is the extension of BTB. For each BTB entry, six more bits are required to record Sentry-Target and Change-Page fields, which consumes additional power. However, compared to branch tag and target fields, these power consumptions are slight. Simulation results show that the power of branch predictor is increased by only 5.5% on average. Note that we did not employ any power-saving approaches for the branch predictor in the experimentation. Another kind of hardware cost in CASA-based IFU is the additional registers among pipeline stages. For each cache line, CASA needs one register to transfer the “Line wakeup” signal from ANALYSIS stage to WAKEUP stage, and another register to transfer the “Wordline access” signal from WAKEUP stage to FETCH stage. These pipeline registers also cause extra power. However, the power dissipation in these pipeline registers could be small because most of them are inactive for a long period until the corresponding cache lines are accessed. Simulation results show that the pipeline registers account for 3.4% power in CASA-based IFU, on average. In all, compared to the power of baseline IFU, the added hardware induces more power consumption by 2.7%, on average.

Although the added hardware consumes some power, due to the power reductions of I-Cache and ITLB, the IFU power is reduced by 59.1%, on average, as illustrated in Fig.13. The power dissipation in each IFU component is shown in Table 3. In fact, I-Cache and ITLB no longer account for a large portion of power in CASA-based IFU. The component which needs more concerns of power reduction becomes the branch predictor, which accounts for 72.8% of power in CASA-based IFU.

As illustrated in Fig.13, the overall power consumption of the processor is reduced by about 9.8% on average. In practice, the power of the processor can be affected by many factors, such as the back-end architectures, processor clock domains, and applications executed by the processor. However, these experimental results still demonstrate that CASA can reduce the processor power effectively.

### Table 3. Percentage of Power Dissipation in Each IFU Component Compared to the Entire IFU Power Consumption

<table>
<thead>
<tr>
<th>Component</th>
<th>Baseline</th>
<th>CASA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch Predictor</td>
<td>29.1%</td>
<td>72.8%</td>
</tr>
<tr>
<td>I-Cache</td>
<td>53.2%</td>
<td>20.4%</td>
</tr>
<tr>
<td>ITLB</td>
<td>17.7%</td>
<td>3.4%</td>
</tr>
<tr>
<td>Pipeline Registers</td>
<td>0</td>
<td>3.4%</td>
</tr>
</tbody>
</table>

6.3 Performance

As mentioned in Subsection 5.1, the processor performance is affected by two opposite factors. In this subsection, we take both sides into consideration.

6.3.1 Delay

To evaluate the delay of some critical paths in CASA-based IFU, we focus on the filtering logic in the ANALYSIS stage and cache hit time in the FETCH stage.

In the ANALYSIS stage, the time to access Sentry-Tag arrays and to compare Sentry-Tag costs approximately 0.26ns, and the AND and OR gates shown in Fig.8 for set and way selections cost 0.10ns. Therefore, the time to select a cache line to access is only 0.36ns. In the meanwhile, the time to access ITLB is approximately 0.40ns, and the logic to enable ITLB lookup is 0.06ns. Therefore, the time to generate physical address is 0.46ns.

In the FETCH stage, the time to access different parts of I-Cache is listed in Table 4. The baseline processor spends 0.55ns fetching instructions from I-Cache arrays. However, in the FETCH stage, CASA
spends only 0.42ns fetching instructions from I-Cache, which is 76% of that in the baseline. As discussed in Subsection 5.1, the reason for the reduced I-Cache hit time is that CASA moves the set decoder and some tag bits to the ANALYSIS stage, which reduces the time to access I-Cache arrays in the FETCH stage.

In all, the critical path in CASA costs 0.46ns. Suppose the baseline processor uses I-Cache hit time as the cycle time, which is 0.55ns, then the ideal improvement of processor frequency can achieve 19.6%. For real processor designs, the delay of IFU is quite a design-specific issue. However, according to these results, CASA shows its effectiveness in reducing both the power of I-Cache and ITLB and the delay of critical paths in the IFU.

By comparison, two-level filter scheme\cite{5} increases cache hit time because it has to access the block buffer and Sentry-Tag arrays before cache data arrays in one cycle. By report, it incurs extra delay by 0.10ns, which increases cache hit time by 18.2% (from 0.55 to 0.65ns). In HoA\cite{6}, the virtual page number comparator accessed before ITLB incurs extra delay by 0.20ns, which increases the delay of address translation by 50% (from 0.40 to 0.60ns). As I-Cache and ITLB usually reside on the processor’s critical paths, the increased delay probably translates into the risk of decreasing the frequency. Compared to the two-level filter cache and HoA, CASA is quite superior in reducing IFU delay.

6.3.2 Simulation Cycles

CASA induces some runtime impact due to larger branch misprediction penalties, as discussed in Subsection 5.1. In theory, suppose we wakeup the I-Cache sets before mispredicted branches are resolved, the number of cycles increased by CASA can be:

\[ T_{inc} = N_b \times R_{misprediction} \times 1 \]

where \( N_b \) is the number of dynamically executed branches in the program, and \( R_{misprediction} \) is the branch misprediction rate. Therefore, the performance impact of CASA depends on both the number of dynamically executed branch instructions and the misprediction rate. The runtime increase for each benchmark program is illustrated in Fig.14. For vortex, the runtime increase is small mainly because of the high branch prediction accuracy. Although the branch prediction accuracy for gap and gcc is lower than that for vortex, since the number of dynamically executed branches in them is only about 5%, their runtime increases are also smaller than 0.5%. The runtime increases are smaller than 1% for other benchmarks except twolf. For twolf, since the number of dynamically executed branches is larger than 10% and the branch misprediction accuracy is 12.6%, the runtime increase achieves 1.3%. On average, CASA increases the simulation cycles by only 0.63% compared to baseline. Considering its effectiveness in reducing I-Cache leakage, such runtime increase is negligible.

![Fig.14. Percentage of runtime increase.](image-url)
sor designers. On the contrary, the largest runtime increase of CASA, which is for twolf, is only 1.3%.

In all, considering both delay and simulation cycle, we can conclude that the ideal performance improvement is 18.9%, which is caused by 19.6% of the ideal frequency increase and 0.63% of runtime increase. The worst case is that the performance is sacrificed by 0.63%, which means the frequency is not increased at all due to the delay of other parts in the processor. For real processor design, the change of performance could be between −0.63% and +18.9%. However, even when the worst case occurs, the performance degradation of CASA is still very slight.

7 Conclusions and Future Work

In this paper, we presented CASA for power-efficient I-Cache and ITLB designs. The contribution of this paper, which originated from this new IFU architecture, removes unnecessary restrictions from traditional IFU architectures and provides larger optimization space for the power-saving approaches.

Based on CASA, for I-Cache power reduction, we proposed to analyze the fetch addresses and wakeup I-Cache lines before instruction fetch. The set and way selections focus on triggering only one cache line rather than the whole bank or sub-bank, which reduces more leakage and dynamic power with less performance impact than [4, 5]. Note that for way selections, we proposed to incorporate Sentry-Tag field of BTB with the Sentry-Target arrays, which is superior to extending the BTB with Way-Pointers[21].

For ITLB power reduction, we proposed to extend the BTB and analyze the change of fetch address carefully, which saves comparable power but induces much less timing overhead than the Hardware-Only Approach in [6].

In all, CASA saves the power consumptions of both I-Cache and ITLB very effectively, reduces the delay of critical paths in IFU, incurs minimal runtime increase, and achieves better scalability.

A variety of important issues can be studied in future work. As the branch predictor is accessed before I-Cache, some power-saving approaches can be employed in CASA to reduce the power of branch predictor[31,32], which further reduces the power of the entire IFU. The power-savings can be further evaluated when trace cache is integrated into the processor model. To further reduce the branch misprediction penalties, multiple lines at the alternative path to the branch prediction outcome can be fetched in CASA-based IFU. In all, it is promising that the CASA architecture proposed in this paper can stimulate further work on architectural solutions to power-efficient IFU designs.

References

Han-Xin Sun is a Ph.D. candidate in the School of Information Science and Technology of Peking University. His research interests include computer architecture, energy-efficient microprocessor design, HW/SW co-design, and microprocessor validation.

Kun-Peng Yang is a Ph.D. candidate in the School of Information Science and Technology of Peking University. His research interests include computer architecture, parallel processing, and integrated circuit design, especially energy-efficient CMP/SMP transactional memory.

Yu-Lai Zhao is a Ph.D. candidate in the School of Information Science and Technology of Peking University. His research interests include computer architecture, optimizations of high-end microprocessors for energy efficiency, performance modeling and evaluation, HW/SW co-design, and System-on-Chip.

Dong Tong is a professor in the School of Information Science and Technology of Peking University. His research interests include computer architecture, storage system, interconnection network, and System-on-Chip. He is the co-founder of the UNITY system architecture. He also led the design of the UNITY microprocessor, SoC chip and IP cores.

Xu Cheng is a professor and Ph.D. advisor in the School of Information Science and Technology of Peking University. His main research fields include high performance microprocessor, System-on-Chip, embedded system, instruction level parallelism, HW/SW co-design and compiler optimization. He is the founder of the Microprocessor R&D Center and the UNITY system architecture. He also led the design of the UNITY system software, the UNITY microprocessor, SoC chip, IP cores and network computer.