Tsinghua Science and Technology

Volume 22 | Issue 3

Article 7

2017

Feedback Cache Mechanism for Dynamically Reconfigurable VLIW Processors

Sensen Hu
High Performance Embedded Computation Lab, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.

Weixing Ji
High Performance Embedded Computation Lab, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.

Yizhuo Wang
High Performance Embedded Computation Lab, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China.

Recommended Citation

Sensen Hu, Weixing Ji, Yizhuo Wang. Feedback Cache Mechanism for Dynamically Reconfigurable VLIW Processors. Tsinghua Science and Technology 2017, 22(3): 303-316.

TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/09 pp303–316
Volume 22, Number 3, June 2017

Feedback Cache Mechanism for Dynamically Reconfigurable VLIW Processors

Sensen Hu, Weixing Ji*, and Yizhuo Wang

Abstract: Very Long Instruction Word (VLIW) architectures are commonly used in application-specific domains due to their parallelism and low-power characteristics. Recently, parameterization of such architectures allows for runtime adaptation of the issue-width to match the inherent Instruction Level Parallelism (ILP) of an application. One implementation of such an approach is that the event of the issue-width switching dynamically triggers the reconfiguration of the data cache at runtime. In this paper, the relationship between cache resizing and issue-width is well investigated. We have observed that the requirement of the cache does not always correlate with the issue-width of the VLIW processor. To further coordinate the cache resizing with the changing issue-width, we present a novel feedback mechanism to "block" the low yields of cache resizing when the issue-width changes. In this manner, our feedback cache mechanism has a coordinated effort with the issue-width changes, which leads to a noticeable improvement of the cache performance. The experiments show that there is 10% energy savings as well as a 2.3% cache misses decline on average achieved, compared with the cache without the feedback mechanism. Therefore, the feedback mechanism is proven to have the capability to ensure more benefits are achieved from the dynamic and frequent reconfiguration.

Key words: reconfiguration; Very Long Instruction Word (VLIW); issue-width; cache; feedback

1 Introduction

The capability of Field Programmable Gate Arrays (FPGA) continues to increase as new devices provide a greater programmable area and faster maximum clock speeds. With the increasing transistor densities of current-days chips, it becomes possible to implement larger and more complex processor designs on FPGAs. In the Application Specific Integrated Circuit (ASIC) aspect, research focuses on the simple microarchitecture which has advantages for silicon manufacturing, cost, and power consumption. On the other hand, Very Long Instruction Word (VLIW) architectures have proven to be useful for embedded applications with abundant Instruction Level Parallelism (ILP)[1]. Additionally, VLIW compilation technology has reached a maturity level to grant the exploitation of high-degrees of ILP in the targeted algorithms so that VLIW processors have recently become popular in application specific fields[2]. Nowadays, this is performed for prototyping purposes in the ASIC processor design path or for the shipment of these processors as softcores on FPGAs.

The energy consumption of VLIW processors, as is well-known, rises with the increasing issue-width due to more Function Units (FUs) utilized, which improves the performance of the system. Moreover, the applications exhibit various ILP and even phases within or across applications. Our key motivation is to utilize the necessary resources only when needed. Hence, the

• Sensen Hu, Weixing Ji, and Yizhuo Wang are with High Performance Embedded Computation Lab, School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: {foresthss, pass, frankwyz}@bit.edu.cn.

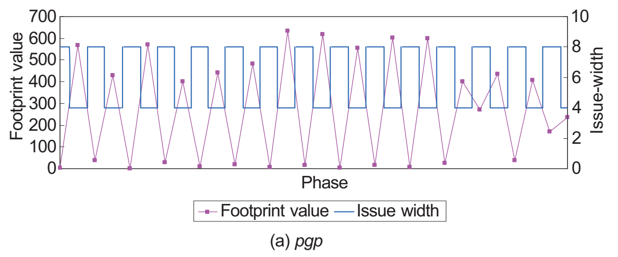
∗ To whom correspondence should be addressed.

Manuscript received: 2016-05-12; revised: 2016-09-27; accepted: 2016-12-23

cache resizing combined with the switching issue-width is an ideal candidate for its performance. On the one hand, it is beneficial for performance to invest issue-width in terms of ILP. On the other hand, it is beneficial for energy consumption to invest cache resources in terms of the utilization of cache resources. In our prior study, we have been able to dynamically reconfigure the issue-width of the VLIW processor and the data cache simultaneously at runtime to match the changes of ILP or phase.

However, a higher demand for the issue-width of the VLIW processor does not always correlate with bigger requirements of cache resources, and vice versa.

Figure 1 illustrates the segments of the footprints for *pgp* and *CRC*32 benchmarks from MiBench[3], in which each benchmark performs to resize the data cache according to the switching issue-width directly. In this paper, the footprint denotes the expected amount of data that a program accesses different cache blocks in a given phase (issue-width), i.e., the number of different locations of cache block accessed. As an important metric, the footprint reflects the requirement of the cache[4]. For the *CRC*32 benchmark, its footprint values appear to follow the same trend in the changes of issue-width, i.e., when staying in the large issue-width, the data cache has a large footprint and vice versa. On the contrary, for the *pgp* benchmark, its footprint values deviate from the changes of issue-width, i.e., its footprint is lower in the large issue-width but higher in the small issue-width. In such a case, cache downsizing leads to the sharp increase of cache misses which possibly has a more detrimental effect on the overall performance. Therefore, it makes sense to "block" such cache resizing rather than the cache always going with the switching of the issue-width based on the utilization of cache.



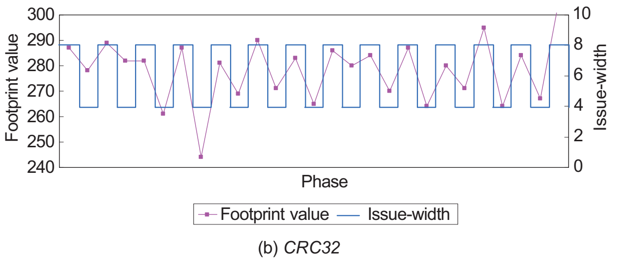


Fig. 1 Segments of the footprints from *pgp* and *CRC32* benchmarks, respectively, are varied when the issue-width was changed separately.

To further improve the performance of the reconfigurable VLIW processor, a feedback mechanism is proposed in this paper, which means a coordinated effort with information from the caches is required to drive the issue-width changes. This proposal ensures more benefits are achieved from the dynamic and frequent reconfiguration. Meanwhile, this design will eliminate the low yields reconfiguration, not only for cores but also for caches. Finally, by using our feedback framework, the significant energy savings can be realized as dynamic changes of issue-width. Compared with the performance of the cache without the feedback mechanism, we achieve 10% energy savings as well as 2.3% cache misses decline on average.

In this paper, we present the following contributions:

- We propose a novel feedback mechanism to perform data cache resizing. Our proposal makes full use of the footprint information to coordinate the switching of issue-width. Further, introducing such a feedback mechanism is able to eliminate low yield cache resizing which reduces the amount of issue-width changes. Consequently, the overall of energy consumption significantly decreases.
- To our best knowledge, we are the first to employ the footprint information to guide cache resizing, which is different from the conventional approach of monitoring cache misses. Moreover, the feedback mechanism further improves the quality of cache resizing. In this manner, we implement the maximum of the benefits of cache resizing.
- The feedback mechanism is implemented in the hardware with little overhead.

The remainder of the paper is organized as follows. The background and our motivations are described in Section 2. In Section 3, the framework of the cache feedback mechanism and the implementation are presented in detail. In Section 4, we describe the evaluation environment. In Section 5, the experimental results are discussed in detail. In Section 6, we present the related work. Finally, we draw our conclusions in Section 7.

2 Motivations and Background

Ideally, for VLIW processors, many instructions can execute in parallel to provide performance improvements. Traditional VLIW processors only have a fixed issue-width. Many researches aim at exploiting the instruction level parallelism of programs to maximize the utilization of all data-paths. However, the diversity of ILP has potential. Instead, in this paper we focus on the coordinated effort between the caches and the issue-width.

2.1 ρ-VEX reconfigurable VLIW processor

The ρ-VEX project[5,6] is developed by computer engineering laboratory of TU Delft (Delft University of Technology), which is a recently introduced VLIW processor design that is capable of dynamically reconfiguring certain aspects of its organization. Figure 2 depicts the dynamic architecture of reconfigurable ρ-VEX processor. The ρ-VEX processor contains a maximum of 8 issue-widths that can be changed at runtime between the 8-issue, 4-issue, or 2-issue modes[7,8]. Different parameters of the ρ-VEX processor, such as the number and type of FUs, the number of multi-ported registers (size of register file), the number and type of accessible FUs per syllable, and the width of memory buses, and different latencies can be dynamically changed at runtime[7]. A key motivation of the ρ-VEX processor design is to utilize only the necessary resources when needed. More specifically, use the 2-issue mode for low ILP programs while using the 8-issue mode for high ILP programs. Consequently, matching the resources to the ILP during runtime can lead to improved resource utilization and reduced energy consumption. Moreover, the split issue-width can be allocated for other applications.

In prior work, Brandon and Wong[10] exploited the generic binaries, i.e., only a single binary needs to be generated to run on the different issue-width core instances. The generic binaries allow us to switch between issue-widths at runtime without the need for check-points. Moreover, the switching between the issue-width modes is performed at runtime and can be achieved in several clock cycles. The target for the generic binaries is an 8-issue machine with 8 ALUs, 4 Multipliers, 1 Load/Store unit, and 1 Branch unit. The layout of the functional units is shown in Table 1. The layouts for 4- and 2-issue versions that support the generic binary are shown in Tables 2 and 3, respectively.

For the VLIW processor, the phase denotes a period between two actual reconfigurations of the issue-width. In the ρ-VEX processor, we apply three strategies to exploit the phase predictors, which improves the performance of the reconfigurable processor.

In terms of the different application specific, three phase predictors[11] can be customized to provide a trade-off between delay and energy, i.e., delay-prior, energy-prior, and Energy-Delay Product (EDP)-prior. For the delay-prior predictor, the issue-width with the shortest delay will be selected, which leads to lower power consumption. For the energy-prior predictor, the issue-width with the smallest energy consumption will be chosen, which results in better performance. For the EDP-prior predictor, the issue-width with the smallest EDP value will be chosen. Such prediction methods are based on the delay and energy information from the past interval used to reconfigure the core for the next

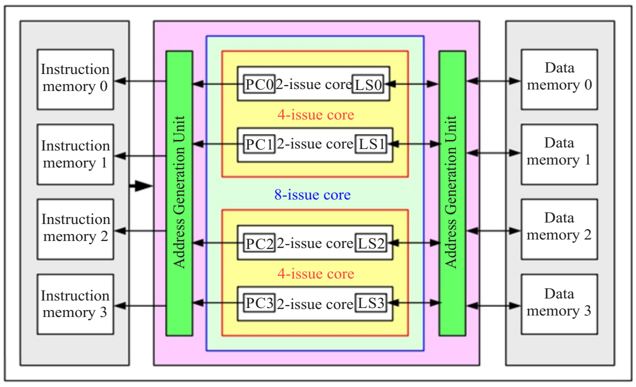


Fig. 2 General view of ρ-VEX processor[9]. The architecture of instruction memories and data memories is parameterized. The ρ-VEX processor is able to reconfigure during run-time in any desired configuration from the biggest version as an 8-issue width to the smallest version as a 2-issue width.

Table 1 Layout of functional units in 8-issue ρ-VEX.

Issue-slot	Functional unit	Issue-slot	Functional unit
0	ALU	4	ALU MEM
1	ALU MUL	5	ALU MUL
2	ALU	6	ALU BR
3	ALU MUL	7	ALU MUL

Table 2 Layout of functional units in 4-issue ρ-VEX.

Issue-slot	Functional unit	Issue-slot	Functional unit
0	ALU MEM	2	ALU BR
1	ALU MUL	3	ALU MUL

Table 3 Layout of functional units in 2-issue ρ-VEX.

Issue-slot	Functional unit	Issue-slot	Functional unit
0	ALU MEM BR	1	ALU MUL

interval.

2.2 Motivations

The ρ-VEX[5,6] platform provides an opportunity to utilize only the necessary resources when needed. As shown in Fig. 2, ρ-VEX is able to execute low-ILP phases on a small cache while high-ILP phases execute on a larger cache. Hence, the inactive issue-width costs little energy. The mentioned switch of issue-width in turn requires a runtime adjustment of the data caches to match the demands from the core(s). The cache adaptation entails upsizing/downsizing the data caches when an external trigger causes a core to use either more or less data paths, respectively.

Cache reconfiguration at runtime affects cache misses, which directly correlates with the performance. As mentioned in Fig. 1, for such a scenario like *pgp*, cache downsizing performs aggressively, causing the losses to outweigh the gains. Therefore, we are motivated to establish a feedback mechanism between the core and data cache by using the footprint information, to make a final decision of cache resizing.

To achieve this goal at a reasonable cost, we need to address the challenge. The footprint varies widely within and across applications. It is ideal to allow a high footprint of data cache in large issue-width while a low footprint in small issue-width. If the requirement of cache resource has the same trend (e.g., *CRC*32 benchmark) as the issue-width, then its performance is not sensitive to the cache resizing. On the contrary, if the requirement of the cache resource is bigger, as the issue-width of the core tends to be smaller (then, data cache downsizing will occur), the performance is highly sensitive to the amount of available cache. Similarly, if the requirement of the cache resource is smaller as the issue-width of core trends to be bigger (then, data cache upsizing will occur), the cache resource is over-provisioned and incurs energy wastage. In such cases, it is important to “block” the cache resizing judiciously by taking footprint information into consideration. Figure 3 depicts the overview of blocking the cache resizing by using feedback, where the top chart shows the cache resizing without feedback and the bottom chart shows the cache with feedback.

Accordingly, we employ a feedback mechanism instead of a slaving cache, which means a coordinated effort is adopted to respond to the issue-width change from the data cache. This feedback mechanism ensures more benefits are achieved from the dynamic and

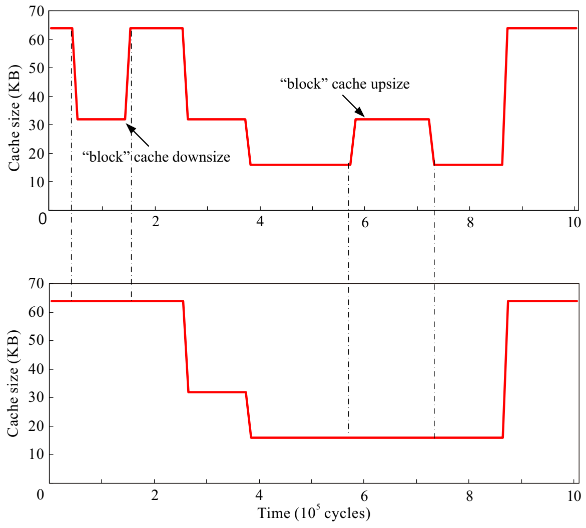


Fig. 3 Overview of feedback blocking the cache resizing: cache resizing without feedback (top) and cache with feedback (bottom).

frequent reconfiguration. Meanwhile, this design will eliminate the low-yield reconfiguration not only for cores but also for caches.

3 Design Specifications

3.1 Framework

Figure 4 depicts the whole framework of ρ-VEX processor with our proposed feedback mechanism. The framework consists of two major parts. In the left of Fig. 4, the delay detector, switching estimator, and phase predictor compose the predictor system of ρ-VEX. In the right of Fig. 4, the monitor, feedback controller, and resizing logic compose the feedback system.

On the initial time, the instruction bundles fetched from the instruction unit are sent to the delay detector and the switching estimator. The basic unit of execution in ρ-VEX is an operation, similar to a typical RISC-style instructions. An encoded operation is called a

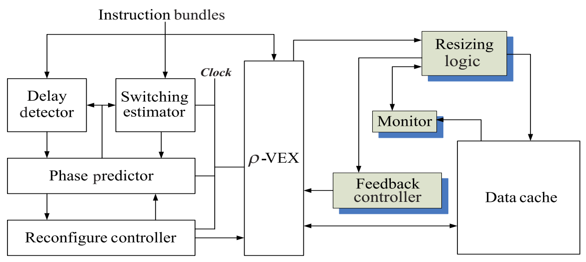


Fig. 4 Framework for the feedback mechanism of ρ-VEX. The shaded parts are newly added to offer feedback mechanism.

syllable, and a collection of syllables issued in a single cycle and executed in VLIW mode as an atomic unit is called an instruction. Note that all the executables are encoded as generic binaries[10]. Our target platform employs the VLIW cores issue instructions as bundles. After a certain time interval, the phase predictor predicts the issue-width for the next execution phase in terms of the information from the delay detector and the switching estimator. If the reconfiguration is needed, the phase predictor sets the *reconfig_enable* signal to inform the reconfiguration controller and then forwards to the resizing logic. Subsequently, the resizing logic looks up the monitor to obtain the footprint information of the data cache and then responds to the request. Once the resizing logic foresees the benefit from such resizing, it confirms the request. Otherwise, the denial response is sent to the reconfiguration controller by the feedback controller, so that the data cache maintains its state without change. Once the confirmation is received, the reconfiguration controller resets the registers in the delay detector and the switching estimator. Finally, the issue-width of the ρ-VEX is reconfigured according to the predicted issue-width as well as the data cache. The resizing logic takes the information predicted for the next issue-width to reconfigure the data cache.

In the maximum of ρ-VEX processor, we employ a baseline 64 KB total size, 32-byte line size, and four-way set-associativity cache, in which the 8-issue, 4-issue, and 2-issue modes have access to a four-way, two-way, and direct-mapped cache, respectively. Hence, cache reconfiguration is based on selective-way[12] in our studies. The selective-way circuit does not appear in critical paths in the cache so that the tags and the data decoders can be operated in parallel, which is a key advantage that allows one to divide the cache more efficient. In particular, the selective-way method has no extra overhead and is able to adapt to our framework.

In addition, the monitor circuit is separated from the data cache, which allows the monitor circuit to obtain utility information about an application for all the ways in the cache. In such ways, the contention is reduced. Note that such feedback delay has no significant impact on the performance because the process is predicted with several hundreds, even thousands of cycles ahead. The prediction system and the feedback module are designed as an integrated hardware module running in parallel to the reconfigurable ρ-VEX processor.

3.2 Cache monitor

In our design, we require a cache monitor to track the number of misses for all possible numbers of cache blocks. To obtain the footprint information from a total size of 64 KB, 4-way data cache (other parameters of the experiment are described in Section 4.2), a straight-forward method is to increase the 4-bit tag directories for each set, denoting each cache block as accessed or not. In such an expensive method, the total 2048 bits are needed due to 512 sets in the data cache. The hardware overhead with tags makes this scheme impractical.

Considering the spatial locality of the application, we use a dynamic set sampling method to monitor the cache. The key idea of this sampling method is that the overall behavior of the cache can be approximated by sampling only a few sets, which reduces the hardware overhead. An important issue to resolve is to determine how many sampled sets are enough to approximate the overall behavior of the cache. In Section 3.3, we present the analytical model.

3.3 Analytical model

We use the Qureshi model[13] to discuss our analytical model. Assume that the cache has m sets. Let F_i denote the number of locations of cache blocks which have been accessed in each cache set i. Accordingly, the mean of the footprint per set (f_1) is computed by

$$f_1 = \sum_{i=1}^{m} F_i/m \tag{1}$$

Let n denote the number of randomly sampled sets from the total m sets. And the number of location of accessed cache blocks in the sampled set is F_i'. The mean of the footprint per set from the sampled n sets is f_2.

$$f_2 = \sum_{i=1}^{n} F_i'/n \tag{2}$$

Consider bounding the value of $|f_2 - f_1|$ to some threshold ε. If the variance σ^2 is in the values of F_i across all the sets, then by Chebyshev's inequality,

$$P(|f_2 - f_1| < \varepsilon) \geqslant 1 - \sigma^2/(n\varepsilon^2) \tag{3}$$

To bound f_2 within one accessed block in f_1, $\varepsilon = 1$.

The probability that f_2 within one accessed block from f_1 is given by

$$\text{Prob} \geqslant 1 - \sigma^2/n \tag{4}$$

Chebyshev's inequality considers only variance without making any assumption about the distribution of the data. For most of the workloads studied, the variance σ^2 is less than 3[13], as few as $n = 32$, the probabilities are sufficient to approximate to 90.6%. This means that the footprint can be obtained by monitoring only 32 cache sets. The baseline data cache in our experiments is 4-way, 64 KB in size, and contains 512 sets. Therefore, we employ a static simple policy, i.e., we select set 0 and every 16th set.

3.4 Hardware implementation

The dynamic set sampling method requires 32 monitors, each with 4 bits. Hence, only a total of 128 bits is needed. Figure 5 illustrates the dynamic set sampling monitor that contains the 4 bits for each cache block. The circuit tracks the footprint of each 16th cache block using the cache hit event. When the cache hit occurs, the corresponding bit is set to 1. Much of the hardware overhead is low. Assume each entry of the cache is 20 bits. The storage overhead of the total monitors is less than 0.024%, in which case the hardware overhead is negligible. Therefore, the sampling method is cost-effective.

3.5 Feedback mechanism

As mentioned above, the feedback mechanism has three sub-modules: the monitor, resizing logic, and feedback controller. (1) The cache monitor is responsible for tracking the specified information of cache utilization in terms of the sample of footprint. (2) The resizing logic involves the resizing algorithm that determines whether to do cache resizing or not. As the reconfiguration parameters of cache are passed to the resizing logic, the cache resizing will occur if the condition of cache resizing is satisfied. Meanwhile, the *resizing-decision* signal generates and passes to the resizing controller. Otherwise, the cache maintains the previous size. (3)

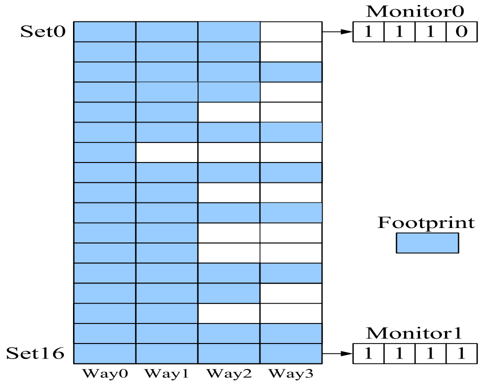


Fig. 5 The footprint monitor with dynamic set sampling.

The feedback controller is responsible for passing the resizing response to the reconfigure controller. If the confirmation signal issued, the reconfiguration controller resets and maintains the previous issue-width. Otherwise, the issue-width will be changed according to the specified request. Figure 6 illustrates our proposed feedback control loop that can be used for the dynamic reconfigure processor.

Algorithm 1 depicts the cache resizing algorithm. The cache first looks up its footprint value the instant the cache resizing request is received. Resizing logic identifies the resizing mode (upsize or downsize).

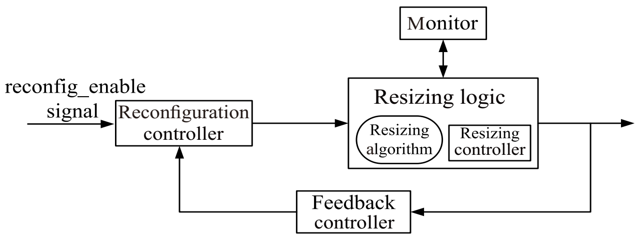


Fig. 6 Illustration of a feedback control loop.

Algorithm 1 Resizing framework with feedback

```
Input: trigger: /*the issue-width mode trigger*/
    Initialization trigger;
    InitialFootprint();
1:  while (1) do
2:    CacheAccess();
3:    UpdateFootprint();
4:    if trigger->flag == True then
5:      footnum = GetFootprint();
6:      if trigger->mode == upsize then
7:        if footnum > threshold_up then
8:          *way = SelectWay(upsize);
9:          way->state = enable;
10:         MergeWay();
11:       else
12:         SendFeedback();
13:       end if
14:     end if
15:     if trigger->mode == downsize then
16:       if footnum < threshold_down then
17:         *way = SelectWay(downsize);
18:         way->state = disable;
19:         SplitWay();
20:       else
21:         SendFeedback();
22:       end if
23:     end if
24:     CleanFootprint();
25:     Reset(trigger);
26:   end if
27: end while
```

Subsequently, the actual cache resizing occurs provided the resizing condition is satisfied. Finally, the monitor is reset after the cache resizing occurs. Otherwise, refused feedback will be sent.

To implement the feedback mechanism, there are a couple of thresholds (threshold_up and threshold_down) that control the strategy of the "to do or not to do" to resize, one threshold that controls whether to upsize the data cache, and the other threshold that controls whether to downsize the data cache. Larger thresholds may cross several phases so as to reduce the opportunities for resizing, while smaller thresholds may weaken the feedback mechanism. More specifically, for the parameter threshold_up, a larger value limits the cache upsizing and keeps the core running in small issue-width for a long time. For the parameter threshold_down, a smaller value limits the cache downsizing and keeps the core running in high issue-width for a long time. Accordingly, the issue-width can be seldom changed. Therefore, both sensitive threshold parameters signify how much cache resizing is permitted.

4 Evaluation

In this section, we present our experimental environment and benchmarks. Subsequently, the results are presented in Section 5.

4.1 Benchmarks

We use 20 benchmarks from the MiBench[3] suite for our studies. The MiBench benchmark suite is widely used to evaluate the embedded system performance. It contains a collection of embedded and portable applications, including automotive and industrial control, consumer devices, office automation, networking, security, and telecommunications. This benchmark suite consists of several stand-alone applications that are easily compiled without the need for porting additional libraries.

Table 4 presents an overview of the benchmarks, in which we list the number of operations, the number of bundles, and the ratio of memory accesses (load and store instructions) to total instrutions for each application of the MiBench benchmark. The statistics listed in Table 4 are from the vex-3.43 compiler (Hewlett-Packard compiler)[14] using the -O3 optimization level and the -fno-xnop -fexpand-div flags.

Table 4 Applications in the benchmark suite.

Benchmark	Number of operations ($\times 10^6$)	Number of bundles ($\times 10^6$)	Ratio of memory accesses (%)
adpcm-rawcaudio	81.7	28.7	41.61
adpcm-rawdaudio	73.6	29.0	36.59
basicmath	1170.7	347.2	28.71
bitcount	69.9	27.0	19.44
blowfish	120.0	49.5	49.08
CRC32	60.0	25.5	52.86
FFT	1708.2	383.7	15.10
gsm-toast	114.7	55.0	50.37
gsm-untoast	58.4	27.1	46.23
jpeg-cjpeg	52.7	21.9	47.54
jpeg-djpeg	20.3	8.5	55.18
pgp	190.1	80.8	40.89
qsort	191.6	79.9	74.34
rijndael	150.2	52.9	57.74
sha	24.0	8.8	57.37
susan	57.4	20.7	50.22
tiff2bw	27.1	11.5	57.71
tiff2rgba	50.9	21.2	56.94
tiffdither	46.6	19.7	38.77
tiffmedian	80.9	35.7	47.19

4.2 Experimental platform setup

Our baseline 8-issue core configuration is shown in Table 5. As explained above, the largest ρ-VEX core has a four-way set-associative data cache in the 8-issue mode. While in the 4-issue mode, the cache is divided over the two cores and therefore also halves the cache size (32 KB, 2-way) for each core. Similarly, in the 2-issue mode, each core has a 16 KB direct-mapped cache. The replacement policy for all cache configurations is Least Recently Used (LRU).

Table 5 System configuration.

Parameter	8-issue core
Processor frequency	37.5 MHz
ALUs	8
MULs	4
Load/Store unit	1
Branch unit	1
L1 I-cache	32 KB, 4-way, 32 byte, 1 cycle latency
L1 D-cache	64 KB, 4-way, 32 byte, 1 cycle latency
Memory access time	10 cycles

Our experimental platform comprises the following elements:

- Cache simulator: We extended the DineroIV[15] cache simulator, which is a sequential trace-driven cache simulator, to be able to simulate the reconfigurable cache as presented in Section 3.
- (Memory) Trace generator: We used a hardware trace unit to generate instruction traces for each benchmark on the FPGA prototype of ρ-VEX. We extracted the memory read and write operations from these traces for use as input to the cache simulator.
- Core phase predictor: As explained in Section 2.1, in this paper, we employ the delay prior predictor[11] which measures the ILP of the benchmark traces and predicts/decides the most suitable mode for the ρ-VEX core to execute according to the delay first. These predictions are utilized to make the mode decisions and are fed to the cache simulator leading to cache resizing. The predictor also outputs the signal to indicate when a resizing decision is pending.
- CACTI: To calculate the dynamic energy consumption in the cache, we utilize CACTI 6.5[16] targeted 90 nm technology. We obtain energy per access for all of the cache configurations with CACTI.
- ρ-VEX prototype: We use an FPGA to prototype the ρ-VEX and run applications on actual hardware. The design runs on a Virtex 6 (ML605 development board) at 37.5 MHz. We use the GRLIB[17] interface with the main memory and other peripherals.

4.3 Methodology

We run all the benchmarks on ρ-VEX to retrieve instruction traces for analysis afterwards. The instruction traces generated on the FPGA are then parsed with a script. This script involves a real instruction sequence of execution. In each instruction bundle, we append an issue-width field which instantly records the issue-width information of ρ-VEX. Hence, the script offers the issue-width scheme. Subsequently, the script is fed to cache simulator who fetches the instructions one by one and processes *load/store* operations. In the first running, cache simulator performs cache resizing according to the extracted issue-width information from the script. In the second running, cache simulator plugs in the feedback mechanism and the prediction function. At each prediction point, the resizing logic assesses whether the cache reconfiguration is needed. As the resizing logic declines the reconfiguration request, the issue-width maintains current mode. Meanwhile, the input script will be revised so as to generate a new order of issue-width after a whole execution.

All the measurements skip the initialization part and warming up of the cache, and the ρ-VEX processor is always initialized to the 8-issue mode. We also simulate the benchmarks by using traditional resizing approach according to the cache misses (the cache uses the default configuration under 8-issue mode).

5 Results

To highlight the effect of the proposed feedback framework, we present the majority of our results which outperform the cache without feedback in this section. We focus on three main metrics: cache miss rate, the number of phase changes, and energy savings. We evaluate our feedback resizing techniques against two previous resizing schemes: (1) the direct resizing by monitoring cache misses, i.e., traditional resizing[18]; (2) the resizing without feedback mechanism according to the issue-width changing.

5.1 Impact on cache miss

Cache downsizing has forced the cache way to be disabled and therefore may increase cache misses, while cache upsizing cannot impact the performance. Our feedback mechanism avoids the detrimental cache resizing occurring in terms of the footprint of the cache. Figure 7 presents the noticeable decline of cache misses with respect to the feedback mechanism compared to those without feedback. It can be observed that 16, among the 20 benchmarks, show the decline of cache misses. More specifically, the cache miss rate with the feedback mechanism is reduced by 2.3% on average. The maximum decline of cache miss rate is 12.1% for *sha*. For *basicmath*, *jpeg-cjpeg*, and *tiff2rgba*, their miss rate decreases are over 4%. On the contrary, for *adpcm-rawdaudio*, *bitcount*, *tiffdither*, and *tiffmedian*, their miss rate only slightly increases because their execution time of small issue-width mode is increased by using the feedback mechanism, as depicted in Fig. 8.

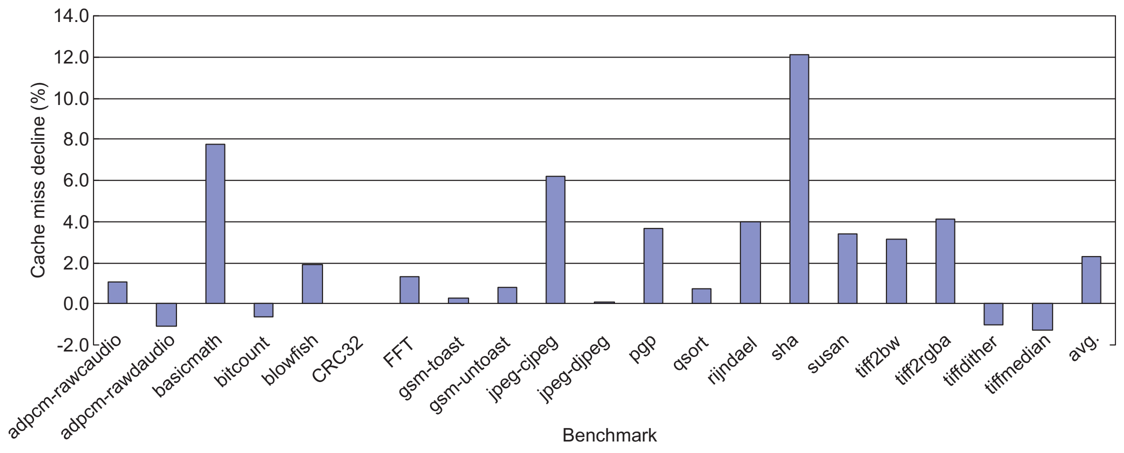


Fig. 7 The decline of cache miss rate with respect to the feedback mechanism compared to those without feedback (normalized to without feedback mechanism).

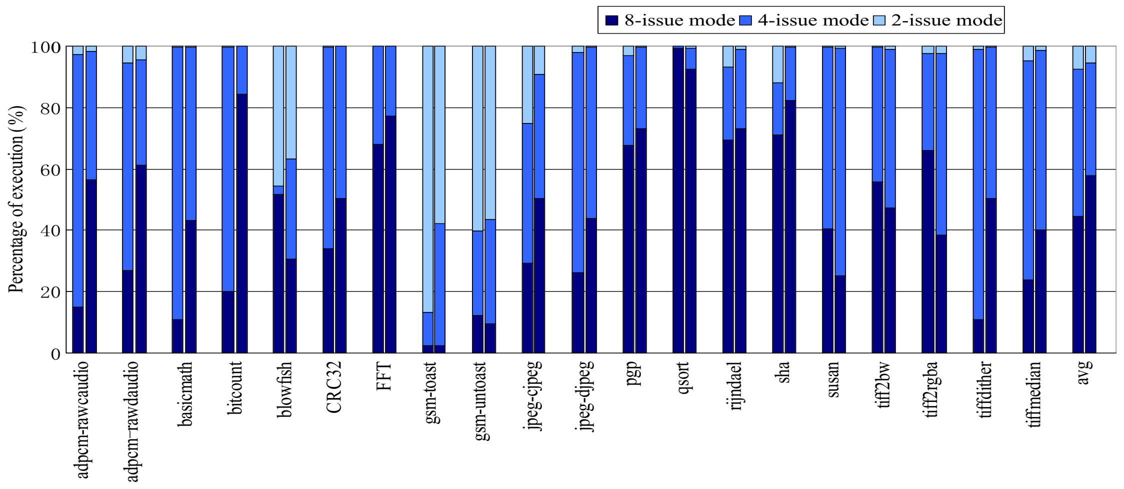


Fig. 8 The percentage execution in each issue-width mode with the feedback mechanism (left bar) compared to that without the feedback mechanism (right bar).

5.2 Impact on phase changes

Table 6 lists the number of phase changes with respect to the feedback mechanism and those without feedback for all benchmarks. Compared with the total number of phase changes without feedback mechanism, the number of phase changes with feedback is significantly reduced by 59.5% on average. For the *qsort* and *blowfish* benchmarks, the decline rate is up to 97.0% and 96.0%, respectively. The minimum reduction of phase changes is more than 11.9% for the *gsm-untoast* benchmark. Therefore, the introduced feedback mechanism can eliminate the low yields reconfiguration so as to maintain efficiently steady issue-width modes at runtime.

To investigate the impact on phase changing, we have an insight into the distribution of execution under different issue-width. As illustrated in Fig. 8, each benchmark has two bars. The left bar shows the percentage of the execution time with our feedback mechanism, while the right bar shows that without the feedback mechanism. Compared with the cache without feedback mechanism, it is obvious that the feedback mechanism makes full utilization of each issue-width. In particular, our approach leads to improvement of execution time in smaller issue-width without any performance degradation. Thereby, our flexible design is proved to utilize only the necessary resources when needed.

5.3 Energy savings

We use CACTI 6.5[16] to model the reconfigurable data cache. Table 7 shows the dynamic energy consumption of each cache access under three different cache

Table 6 Total number of phase changes.

Benchmark	Total number of phase changes with feedback	Total number of phase changes without feedback	Decline rate (%)
adpcm-rawcaudio	375	1352	72.3
adpcm-rawdaudio	565	1473	61.6
basicmath	4148	21 596	80.8
bitcount	18	32	43.8
blowfish	157	3893	96.0
CRC32	246	362	32.0
FFT	1376	5802	76.3
gsm-toast	175	1405	87.5
gsm-untoast	408	463	11.9
jpeg-cjpeg	166	489	66.1
jpeg-djpeg	155	441	64.9
pgp	138	696	80.2
qsort	18	592	97.0
rijndael	969	1338	27.6
sha	18	103	82.5
susan	45	80	43.8
tiff2bw	189	254	25.6
tiff2rgba	383	469	18.3
tiffdither	124	519	76.1
tiffmedian	260	484	46.3
avg.	496.7	2092.2	59.5

Table 7 Dynamic energy consumption of each cache access under three different configurations.

Cache config	Energy consumption (nJ)
64 KB/4-Way	0.268
32 KB/2-Way	0.128
16 KB/1-Way	0.069

configurations.

The cache energy E_{cache} is composed of static energy E_{s} and dynamic energy E_{d}, as shown in Eq. (5). In Eq. (5), i ranges from 1 to 3, which represents the data cache configuration in the 2-issue mode (16 KB/direct-mapped), 4-issue mode (32 KB/2-way), and 8-issue mode (64 KB/4-way), respectively.

$$E_{\text{cache}} = \sum_{i=1}^{3}(E_{\text{s}_i} + E_{\text{d}_i}) \tag{5}$$

The static energy consumption, E_{s}, is computed by

$$E_{\text{s}} = \text{Cycles} \times \text{energy/cycle} \tag{6}$$

where "Cycles" is the total number of cycles for the benchmark to execute, and "energy/cycle" is the total static energy consumed per cycle. However, low-power CMOS research has typically focused on dynamic power, under the assumption that static energy is a small fraction, only accounting for less than 10% of the total energy[19, 20]. Hence, we only consider the dynamic energy consumption. The dynamic energy consumption, E_{d}, is computed by

$$E_{\text{d}} = (\text{Hit} + \text{Miss} \times \text{Kmiss}) \times \text{energy/access} \tag{7}$$

where the factor "energy/access" is the cache energy consumed by the cache itself plus the extra energy from the reconfiguration-related logic (including extra wires and gates, and a way mask register). "Kmiss" is a factor representing a multiple of the cache hit energy consumption. According to Ref. [20] which considers memory hierarchy including the external memory, the value of "Kmiss" ranges from 50 to 200. For this paper, we assume "Kmiss" to be 50. "Hit" and "Miss" denote the number of hits and the number of misses for the corresponding cache configuration in different issue-width modes, respectively.

Figure 9 depicts the energy savings of the data cache. Figure 10 illustrates the average data cache size achieved by each mechanism. All the results in two figures are normalized to those without the

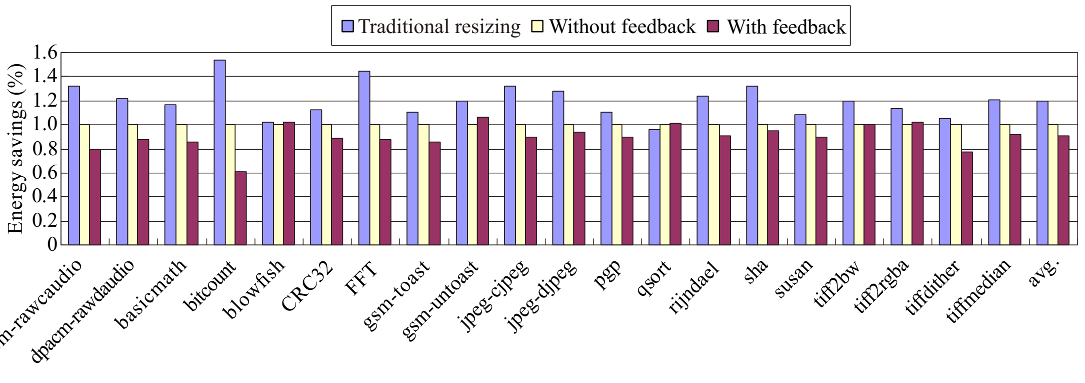


Fig. 9 The energy savings with respect to the feedback cache compared to that without feedback (normalized to without feedback mechanism).

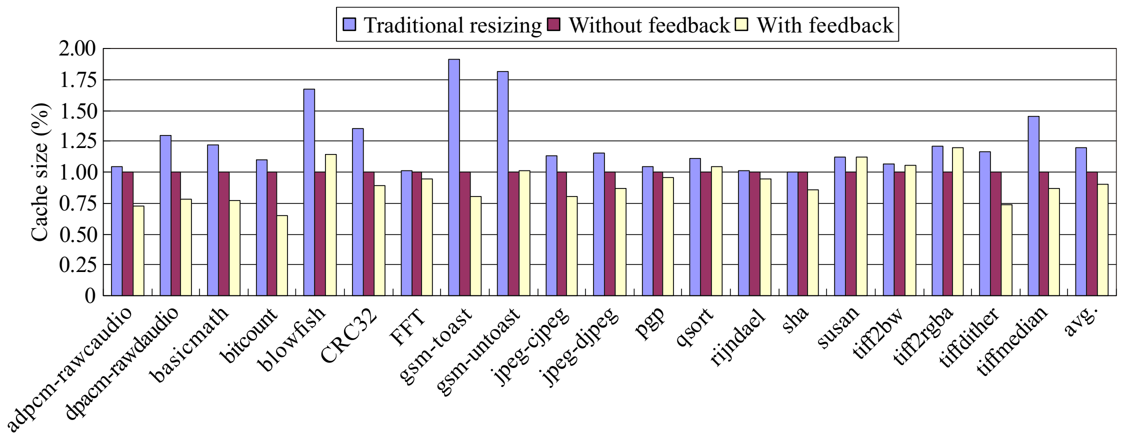


Fig. 10 The average cache size with respect to the feedback cache compared to that without feedback (normalized to without feedback mechanism).

feedback mechanism. As we can see from Figs. 9 and 10, our feedback resizing technique shows noticeable improvements on average over the traditional resizing and that without feedback in terms of cache size or energy consumptions. Compared to the cache without feedback, it is obvious that the feedback cache achieves 10% energy savings on average. The cache with feedback outperforms the cache without feedback in 16 benchmarks. It is also true that the average cache size is reduced by 9.7% on average. For the *adpcm*-rawcaudio, *bitcount*, *jpeg-cjpeg*, and *tiffdither* benchmarks, they achieve significant cache size decline as well as energy savings. We also observe that both energy consumption and cache size increase for the *blowfish*, *qsort*, *tiffi2bw*, and *tiff2rgba* 4 benchmarks. The reason for this phenomenon is that the benchmarks prefer the higher issue-width mode due to the introduced feedback mechanism, as illustrated in Fig. 8. (Note that the size of the data cache is fixed if we do not include the issue-width changes to guide the cache reconfiguration, i.e., 64 KB in the 8-issue mode or 32 KB in the 4-issue mode, or 16 KB in the 2-issue mode.)

The leakage of cache (static energy consumption) is also reduced due to the utilization of the cache size. Considering this factor and the declining number of actual issue-width changes, the cache with feedback can reach more significant energy savings.

6 Related Work

6.1 Reconfigurable processor

Reconfigurability means that one can modify the functionality of a reconfigurable device according to the requirement in execution, which is able to bridge the gap between the processors and ASICs in terms of flexibility and performance[21]. For this reason, many reconfigurable systems have been introduced. As explored in the previous designs, the reconfigurable fabric choices are chiefly restricted among the FUs of the processor, which is particularly complex, not only for the developer but also for the customer.

For example, ADRES[22] provided a VLIW model for the base processor with design exploration options; whereas, in KAHRISMA[23], its fabrics can be combined to realize different instruction set architectures. In addition, the NIOS II[24] soft-processor, Xtensa[25], and ARC[26] configurable processors provided the user with the flexibility to utilize both custom instructions and co-processors to accelerate the execution of an application. Moreover, Xtensa was supported by a development framework capable of automatically identifying custom instructions.

The commonly used commercial VLIW processors such as the TriMedia[27] series from NXP, ST231[28] from STMicroelectronics, TMS320C611[29] from Texas Instruments, and Crusoe[30] from Transmeta, all utilized a fixed issue-width. However, reconfiguring the issue-width at runtime improves the performance of applications with various ILP. The ρ-VEX develops a VLIW processor that can dynamically switch issue-width between the 8-issue, 4-issue, and 2-issue modes. In this paper, our work targets this VLIW processor. All the applications of the benchmark are compiled to generic binaries, and then executed in the ρ-VEX platform.

6.2 Cache resizing

The statistics of energy consumption shows that the on-chip cache consumes about 16% of the total power in Alpha 21264, 30% in StrongARM, and 24% in Niagara-2[31,32]. The power consumption of the cache is becoming a large fraction of the processor power, which is increasing with the growth of the cache size. Hence, the cache plays an important role in the performance and the energy consumption optimization.

Cache resizing determines the best cache configuration with respect to one or more design metrics, such as total size, line size, associativity, etc. For example, if the cache size is larger than the working set size, excess dynamic energy is consumed by fetching blocks from an exceptionally large cache, and excess static energy is consumed to power the large cache size. Alternatively, if the cache size is smaller than the working set size, excess energy is wasted due to thrashing[33].

Previous studies on the granularity of cache resizing were diverse. Some work focused on the cache way, set, and line. In Refs. [34–40], the authors employed a way-resizing method which allows for varying the set-associativity. In Refs. [18, 41], the authors employed a set-resizing method which varies the number of cache sets. In Refs. [12,18,42], the authors employed a hybrid method, i.e., set-and-way resizing.

Our work, however, differs from these work in several significant ways. First, in contrast to triggering the cache resizing event, our work suggests using the issue-width switching instead of monitoring the cache misses. Second, we perform a feedback mechanism that negotiates with the core. The cache has the capability to refuse the decision from the core and maintain itself when the cache identifies the low yields resizing.

7 Conclusion

As a reconfigurable softcore VLIW processor, the ρ-VEX processor presents unique dynamic features that are well-suited for embedded applications. Its issue-width mode changes allow for much better core resource utilization, but it requires an adaptive mechanism to follow suit. To further coordinate cache resizing with the issue-width changing, we present a novel feedback mechanism to perform data cache resizing. To our best knowledge, we are the first to attempt to "block" the low yields of cache resizing using the information of the cache footprint, which is different from the conventional approach which monitors the cache misses. In this manner, we not only implement the maximum of the benefits of cache resizing, but also provide a trade-off between ILP and Data Level Parallelism (DLP). Further, our reconfigurable cache with feedback mechanism is proven to have the capability to maintain a low miss rate to ensure benefits are achieved from the dynamic and frequent reconfiguration. Thereby, our feedback cache mechanism uses a coordinated effort with the issue-width changes, which leads to a noticeable improvement in the cache performance. Lastly, we demonstrate that by using our feedback framework, significant energy savings can be realized. Compared to the performance of the cache without the feedback mechanism, we achieve 10% energy savings as well as the significant 2.3% cache misses decline, on average. The outcome of this work can be used to guide the dynamic reconfiguration of components in embedded reconfigurable systems.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 61300010 and 61300011). The authors would like to thank the China Scholarship Council (CSC) for their financial support and Prof. Stephan Wong for his valuable suggestions and kind help.

References

[1] J. Lee, J. M. Youn, D. Cho, and Y. Paek, Reducing instruction bit-width for low-power vliw architectures, *ACM Trans. Des. Autom. Electron. Syst.*, vol. 18, no. 2, pp. 1–32, 2013.

[2] A. K. Jones, R. Hoare, D. Kusic, J. Fazekas, and J. Foster, An fpga-based vliw processor with custom hardware execution, in *Proceedings of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2005, pp. 107–117.

[3] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, Mibench: A free, commercially representative embedded benchmark suite, in *2001 IEEE International Workshop on Workload Characterization*, 2001, pp. 3–14.

[4] X. Xiang, C. Ding, H. Luo, and B. Bao, Hotl: A higher order theory of locality, in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA, 2013, pp. 343–356.

[5] S. Wong, T. Van As, and G. Brown, ρ-vex: A reconfigurable and extensible softcore VLIW processor, in *International Conference on ICECE Technology*,

IEEE, 2008, pp. 369–372.

[6] F. Anjam, S. Wong, L. Carro, G. L. Nazar, and M. B. Rutzig, Simultaneous reconfiguration of issue-width and instruction cache for a VLIW processor, in *2012 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS XII*, Samos, Greece, 2012, pp. 183–192.

[7] F. Anjam, M. Nadeem, and S. Wong, Targeting code diversity with run-time adjustable issue-slots in a chip multiprocessor, in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2011, pp. 1–6.

[8] F. Anjam, M. Nadeem, and S. Wong, A VLIW softcore processor with dynamically adjustable issue-slots, in *2010 International Conference on Field-Programmable Technology (FPT)*, 2010, pp. 393–398.

[9] F. Anjam, Run-time adaptable VLIW processors: Resources, performance, power consumption, and reliability tradeoffs, Tech. Rep., TU Delft, Delft University of Technology, The Netherlands, 2013.

[10] A. Brandon and S. Wong, Support for dynamic issue width in VLIW processors using generic binaries, in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2013, pp. 827–832.

[11] Q. Guo, A. Sartor, A. Brandon, A. C. S. Beck, X. Zhou, and S. Wong, Run-time phase prediction for a reconfigurable VLIW processor, in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1634–1639.

[12] G. Keramidas and C. Datsios, Revisiting cache resizing, *International Journal of Parallel Programming*, vol. 43, no. 1, pp. 59–85, 2015.

[13] M. K. Qureshi and Y. N. Patt, Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches, in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39*, Washington DC, USA, 2006, pp. 423–432.

[14] Hewlett-Packard, http://www.hpl.hp.com/downloads/vex, 2015

[15] M. Hill and J. Edler, Dineroiv trace-driven uniprocessor cache simulator, http://www.cs.wisc.edu/markhill/DinerIV, 2015.

[16] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, Cacti 6.0: A tool to model large caches, Tech. Rep., HP Laboratories, 2009.

[17] GRLib, http://www.gaisler.com/index.php/products/ipcores/soclibrary, 2016.

[18] S.-H. Yang, M. D. Powell, B. Falsafi, and T. Vijaykumar, Exploiting choice in resizable cache design to optimize deep-submicron processor energy-delay, in *Proceedings of Eighth International Symposium on High-Performance Computer Architecture*, 2002, pp. 151–161.

[19] B. Egger, J. Lee, and H. Shin, Dynamic scratchpad memory management for code in portable systems with an MMU, *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 2, pp. 1–38, 2008.

[20] C. Zhang, F. Vahid, and R. Lysecky, A self-tuning cache architecture for embedded systems, *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 2, pp. 407–425, 2004.

[21] R. Hartenstein, A decade of reconfigurable computing: A visionary retrospective, in *Proceedings of the Conferene on Design, Automation and Test in Europe*, 2001, pp. 642–649.

[22] B. Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins, Architecture exploration for a reconfigurable architecture template, *IEEE Design Test of Computers*, vol. 22, no. 2, pp. 90–101, 2005.

[23] R. Koenig, L. Bauer, T. Stripf, M. Shafique, W. Ahmed, J. Becker, and J. Henkel, Kahrisma: A novel hypermorphic reconfigurable-instruction-set multi-grained-array architecture, in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010, pp. 819–824.

[24] NOIS, http://www.altera.com/products/ip/processors/nios2, 2016.

[25] Xtensa, http://ip.cadence.com/ipportfolio/tensilicaip/xtensa-customizable, 2016.

[26] ARC, Arc tangent processor, http://www.arc.com, 2016.

[27] J.-W. van de Waerdt, S. Vassiliadis, E. Bellers, and J. Janssen, Motion estimation and temporal up-conversion on the tm3270 media-processor, in *2006 Digest of Technical Papers International Conference on Consumer Electronics*, 2006, pp. 315–316.

[28] P. Faraboschi, G. Brown, J. Fisher, G. Desoll, and F. Homewood, Lx: A technology platform for customizable vliw embedded processing, in *Proceedings of the 27th International Symposium on Computer Architecture*, 2000, pp. 203–213.

[29] Texas Instruments, http://www.ti.com/lit/ug/spru131g/spru131g.pdf, 2016.

[30] Transmeta, Crusoe exposed: Reverse engineering the transmeta tm5xxx architecture ii, Tech. Rep., Realworldtech.com, 2004.

[31] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures, in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[32] K. A. Vardhan and Y. N. Srikant, Exploiting critical data regions to reduce data cache energy consumption, in *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*, New York, NY, USA, 2014, pp. 69–78.

[33] W. Zang and A. Gordon-Ross, A survey on cache tuning from a power/energy perspective, *ACM Comput. Surv.*, vol. 45, no. 3, pp. 1–49, 2013.

[34] J. Abella and A. González, Heterogeneous way-size cache, in *Proceedings of the 20th Annual International Conference on Supercomputing*, New York, NY, USA, 2006, pp. 239–248.

[35] C. Zhang, F. Vahid, and W. Najjar, A highly configurable cache architecture for embedded systems, in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on, IEEE*, 2003, pp. 136–146.

[36] D. H. Albonesi, Selective cache ways: On-demand cache resource allocation, in *Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on, IEEE*, 1999, pp. 248–259.

[37] J. C. Ku, S. Ozdemir, G. Memik, and Y. Ismail, Thermal management of on-chip caches through power density minimization, in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 38*, Washington DC, USA, 2005, pp. 283–293.

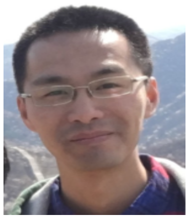
[38] P. Ranganathan, S. Adve, and N. P. Jouppi, Reconfigurable caches and their application to media processing, in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, New York, NY, USA, 2000, pp. 214–224.

[39] N. Mohyuddin, R. Bhatti, and M. Dubois, Controlling leakage power with the replacement policy in slumberous caches, in *Proceedings of the 2nd Conference on Computing Frontiers*, New York, NY, USA, 2005, pp. 161–170.

[40] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli, Exploiting temporal locality in drowsy cache policies, in *Proceedings of the 2nd Conference on Computing Frontiers*, New York, NY, USA, 2005, pp. 371–377.

[41] S.-H. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. Vijaykumar, An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance i-caches, in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on, IEEE*, 2001, pp. 147–157.

[42] S. Mittal and Z. Zhang, Encache: Improving cache energy efficiency using a software-controlled profiling cache, presented at IEEE International Conference on Electro/Information Technology, Indianapolis, IN, USA, 2012.



Sensen Hu received the MS degree from Southwest University of Science and Technology in 2010. He currently works as a PhD candidate at Beijing Institute of Technology. In 2014, he earned the financial support from China Scholarship Council (CSC) to Delft University of Technology as a joint PhD student. His research interests include cache optimization and multi-core architecture.

Weixing Ji received the PhD degree in computer science from Beijing Institute of Technology in 2008. He is currently an associate professor with the School of Computer Science, Beijing Institute of Technology. His research interests include computer architecture and parallel computing.

Yizhuo Wang received the PhD degree in computer science from Beijing Institute of Technology in 2005. He is currently a lecturer with the School of Computer Science, Beijing Institute of Technology. His research interests include computer architecture and parallel computing.