Journal of Computers Vol. 28, No. 3, 2017, pp. 60-78
doi:10.3966/199115592017062803006

Dynamic Relocation Cache for Instruction Delivery in Low Power Processor

Meng-Rao Tang[1], and Hong-Yin Luo[2*]

[1] Department of Electronic Information Engineering, Xiamen Institute of technology, Xiamen, Fujian 361024, People's Republic of China
MRtang@xit.edu.cn

[2] Fujian Newland Computer Co., Ltd, Fuzhou, Fujian 350015, People's Republic of China
luohy@newlandcomputer.com

Received 10 May 2016; Revised 12 August 2016; Accepted 12 December 2016

Abstract. A set-associative cache wastes power because the parallel access to multi-bank memory consumes a lot of power. In this paper, we present a cache architecture (Dynamic Relocation (DR) Cache) that serves as a low-power instruction source instead of the set-associative cache. Not restricted to the static layout imposed by compiler, DR cache is capable of storing instructions in an execution sequence by using a hardware-only method without software or compiler. The trace-based storing scheme, which is capable of storing instructions in an execution sequence, makes sure that DR cache could provide a high hit rate with a small single-bank data memory. We evaluate DR cache in runtime performance and power, and then compare it with the following caches: direct-mapped, 2-way set-associative and 4-way set-associative cache. The comparison is accomplished by running ten embedded programs on a RTL (Register Transfer Level) hardware model based on the LEON3 processor. The evaluation shows that, on average, the 4-kB DR cache provides the same performance in hit rate and an 83% reduction in power consumption compared to the 4-kB 4-way set-associative cache. The 4-kB DR cache also surpasses other caches in what we define as follows: (1) the power with comparable area, and (2) the smallest power.

Keywords: address-mapping, cache, dynamic relocation, low power, processor

1 Introduction

In embedded processors, reducing the power of a memory subsystem has attracted great interest recently [1]. The first-level memory, which is faster and more energy efficient than other level memories in memory hierarchy, has the following two entirely different schemes, namely, the hardware-based cache and the software-based scratchpad memory (SPM). Being transparent to software programs, cache can boost its performance by simply increasing the memory capacity at the cost of extra on-chip area and energy consumption. On the other hand, the SPM controlled by software can improve its performance by using an advanced compiler.

The conception of cache was proposed early in the 1960's [2]. Since then, cache has been extensively used in almost all computers from the first announced mainframe computer IBM 360/85 [3] to the up-to-date embedded processor ARM Cortex-A9 [4]. The well-known set-associative cache [5], comprehensively investigated in the 1980's, has been widely adopted because of its excellent runtime performance. However, because of the parallel access to multibank memory, the set-associative cache wastes power. In order to improve the energy efficiency of cache, some novel caches, such as filter cache [6-7], reused cache [8-9] and trace cache [10], have been proposed. The basic idea of these caches is

* Corresponding Author

identical, using a small level-zero cache memory to store the most frequently accessed data. Unfortunately, these power-saving techniques occupy extra on-chip areas and induce extra delays when a miss happens.

The energy-efficient SPM was proposed for low power purpose in this decade [11-12], and could be used without a source code [1]. However, in order to optimize the memory access pattern of SPM globally, all codes must be input into a special compiler simultaneously. Therefore, it is difficult for SPM to deal with some complicated circumstances, such as the operating systems with scalable application environment. Because of this limitation, SPM is useful for a simple embedded system, but cannot replace cache in all circumstances.

The idea of the dynamic relocation scheme is recently proposed [13]. This paper presents the detail architecture of the DR cache and the comprehensive evaluation. The DR cache is an instruction source for replacing traditional set-associative cache in all circumstances. The DR cache, composed of a data memory and an address-mapping module, is a good candidate for the low power cache. It uses one single-bank data memory to store instructions without tag memory, which is similar to SPM. Furthermore, the instructions with compiler-assigned address is reassigned into a physical location of data memory according to its execution sequence, reducing the collision-induced miss [14] and providing high hit rate.

For evaluating power consumption and runtime performance as accurately as possible, we use a RTL (Register Transfer Level) hardware model based on the LEON3 processor [15] to measure various performance metrics. The runtime performance (IPC and cache hit rate) is measured by running ten embedded applications under the Mentor Modelsim simulation environment. A set of performance monitors embedded in the processor record the data for obtaining the runtime performance. Furthermore, the power consumption is estimated by a Synopsys Design Compiler [16] with the switch activity information that is obtained during the simulation process.

The DR cache is compared with the direct-mapped cache, the 2-way set-associative cache, and the 4-way set-associative cache that is defined as the reference case. On the average, the 4-kB DR cache provides the same performance in hit rate and shows an 83% reduction in power consumption compared to the 4-kB 4-way set-associative cache. The 4-kB DR cache also surpasses other caches in what we define as follows: (1) the power with comparable area, and (2) the smallest power.

The rest of this paper is organized as follows: Section 2. This section lists the related work. Section 3. This section describes the architecture of the DR cache, the dynamic relocation scheme and the address-mapping module. Section 4. This section analyzes the power consumption of caches. Section 5. This section describes the evaluation methodology. Section 6. This section describes the evaluation results and presents the discussion. Section 7. This section concludes this paper.

2 Related Work

2.1 Scratchpad Memory

Scratchpad memory (SPM) is a single data memory without any hardware control logic. The instruction and data are allocated into SPM through two schemes: the static and the dynamic. In the static allocation scheme, Avissar et al. [11] used knapsack formulation and ILP solver to find out the frequently used data object and program routings, and then stored them into SPM permanently. In dynamic allocation scheme, Steinke et al. [17] used a function call to copy a set of instructions and data into SPM dynamically. The candidates that will get into SPM are selected by analyzing the source code and solving an ILP problem. Egger et al. [18] proposed a dynamic allocation technique that just requires object code and binary code. Furthermore, the code and data that will be allocated into SPM is selected by an ILP solver. Then, an exception mechanism is used to copy there codes into the SPM.

The SPM is power-efficient since it has only one single-bank memory without any other hardware control logics. The hit rate is guaranteed by the compiler. This scheme moves the on-chip power consumption to the host compiling system. The DR cache with one single-bank data memory is also power-efficient. However, the high hit rate in the DR cache is guaranteed by the trace-based storing scheme that is a pure hardware scheme.

2.2 Filter Cache

Kin et al. [19] proposed the filter cache and indicated its low-power advantage in 1997. Essentially, the filter cache is a small level-zero cache that consumes less power than the level-one cache. The power consumption can be reduced by increasing the hit rate of this level-zero cache. In order to increase the hit rate, Bellas et al. [20] stored the loop code into the filter cache with the support of compiler. Janapsatya et al. [7] used an efficient replacement policy to keep the frequently accessed instruction in the filter cache. Furthermore, because this extra cache architecture induces the extra delay when a miss happens, a prediction scheme [21] was proposed to mitigate this performance degradation.

In the DR cache and filter cache, the key idea of power reduction is almost same: Using a small memory as the instruction source. However, the scheme is entirely different. The DR cache uses dynamic address-mapping scheme, while the filter cache still uses the static address-mapping scheme. In addition, the DR cache provides the low-power feature without inducing extra clock cycle when a miss happens.

2.3 Reuse Cache

The reuse cache was proposed for storing recycling instructions. It can be used in instruction or trace granularity. In the instruction granularity, Sodani et al. [22] proposed a scheme to reuse repeated computation results, and investigated the possibility of instruction reuse. In the trace granularity, Charles et al. [23] proposed a trace-reused scheme for improving the ILP (instruction level parallel) of superscalar processor. In embedded processor, Tsai et al. [1] proposed a trace reuse cache to be an alternative source for instruction delivery. The retired instructions from the pipeline back-end of processor will be reused for the prediction purpose.

The DR cache also stores the trace of the instructions. However, it has different logic location compared with the reuse cache. Normally, the reuse cache works as an assistant structure. The set-associative cache is still available for the most instruction delivery. However, the DR cache is proposed to instead of the set-associative cache, being the primary instruction source. Furthermore, the trace address in DR cache is calculated by an address-mapping module, but not the lookup table that has been used in [1].

2.4 Trace Cache

In superscalar processors, the fetch bandwidth is a bottleneck for the high runtime performance, so Rotenberg et al. [9] proposed a trace cache to break this bottleneck. The trace cache works as an assistant structure to provide the continuous basic blocks. The key idea of the trace cache is rearranging the instructions in execution sequence instead of the static layout provided by compiler. For reducing power consumption, Hu et al. [24] proposed a prediction scheme to predict the fetch direction.

The key idea of trace cache that rearranges the instructions in execution sequence inspired this work. Although the DR cache and the trace cache have almost same trace-based scheme, they are different in the basic usage. The DR cache is proposed for the low power processor, but the trace cache was proposed for superscalar processor to increase the fetch bandwidth. Furthermore, the DR cache can replace the set-associative cache as a primary instruction source, but the trace cache is used as an assistant structure to provide the instruction in some special condition.

3 Dynamic Relocation Cache

A dynamic relocation cache is proposed as a low-power instruction source for replacing the set-associative cache in any circumstances. The DR cache contains the following three modules: (1) the controller, (2) the data memory, and (3) the address-mapping module. There are two addresses for instructions -- the compiler address assigned by the compiler and the trace address assigned by an address-mapping module according to the instruction execution sequence. To provide a high hit rate, the trace of instructions is stored in the data memory. On the other hand, to capture the trace, instructions with the compiler address are reassigned a trace address by an address-mapping module. This trace address is further used in the logic operations of the address-mapping module. In this section, we

describe the architecture of the DR cache, the dynamic relocation scheme, and the address-mapping module.

3.1 Architecture

The DR cache, which is proposed as the first-level memory for instruction delivery, has the same behavior and interface as a traditional set-associative cache. The block diagram of the DR cache is illustrated in Fig. 1. The DR cache consists of three parts: the controller, the data memory, and the address-mapping module. The controller coordinates the operations of all modules in the DR cache, and communicates with the main memory and the instruction pipeline. The data memory, which is used to store instructions, is identical to the one in the set-associative cache. The address-mapping module, which is used to store the tag address within the compiler addresses of instructions, has a similar function with the tag memory of the set-associative cache.

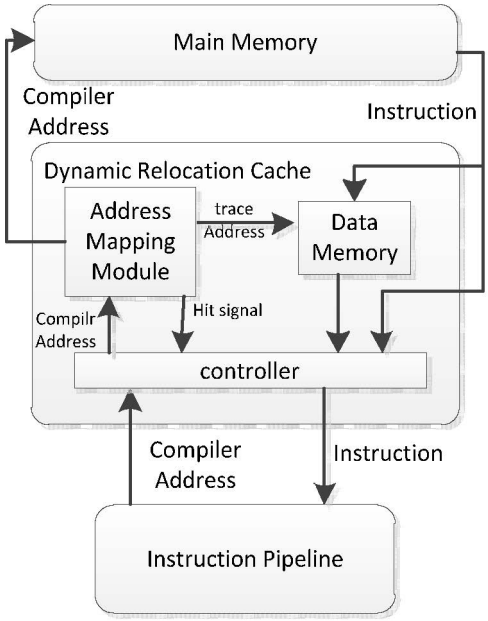

Fig. 1. Architecture of dynamic relocation cache for instruction delivery

After forwarding a compiler address to the DR cache, the instruction pipeline receives an instruction from the DR cache (or the main memory) depending on the hit signal that is provided by the address-mapping module. On the other hand, the main memory receives a compiler address from the DR cache only when a cache miss occurs. Then the main memory provides the corresponding instruction to the DR cache and the instruction pipeline separately. Furthermore, the address-mapping module maps the compiler address to the trace address, and provides a hit signal.

3.2 Dynamic Relocation Scheme

The Dynamic relocation in the DR cache is an address-mapping scheme. The address-mapping scheme defines how the instructions are stored in the data memory. The instructions have the following two addresses: compiler address and trace address. In the set-associative cache, the physical location of instructions in data memory is specified directly by the low n-bit of this compiler address. Since the compiler address of instructions is assigned by the compiler and is unchangeable, the address-mapping scheme of the set-associative cache is static. One instruction has only a few specific locations to be stored, one in main memory and the ‘n’ others in the data memory of n-way set-associative cache.

This static address-mapping scheme has the advantage of easy implementation, but it also can cause the collision-induced miss [14] that is illustrated in Fig. 2. As shown in Fig. 2, the instruction “A” with compiler address 0x0100 and the instruction ‘F’ with compiler address 0x1100 both should be stored in the physical location of cache data memory with address 0x100 inducing the cache miss if this cache has

only one-way data memory. However, in multi-way set-associative architecture (usually implemented by multi-bank tag and data memory) these collisions can be avoided effectively because the “A” and “F” can be stored in a different bank of data memory.

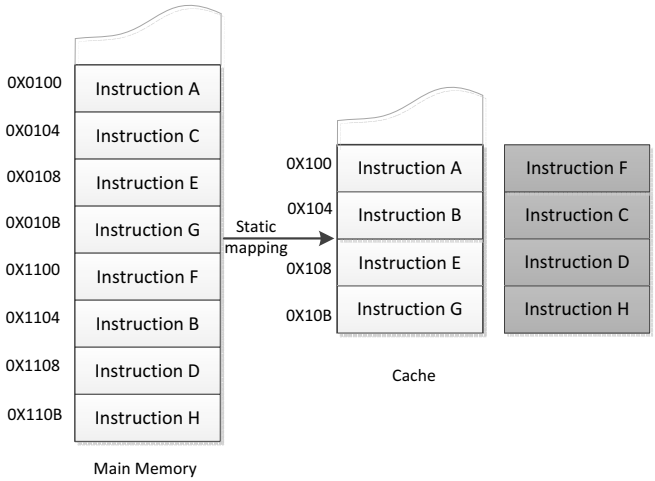

Fig. 2. Static address-mapping scheme in traditional set-associative cache

On the other hand, the DR cache does not use the compiler address to specify the physical location of instructions in the data memory. Instead, the dynamic relocation scheme is used to assign a new trace address for the incoming instruction according to the instruction execution sequence. This trace address is completely different and unrelated to the compiler address, and can be reassigned according to the instruction execution sequence. Because the trace address of instruction can be dynamically changed and one instruction can be stored in any location of the data memory of the DR cache, the address-mapping scheme in the DR cache is dynamic. Fig. 3 shows one example of the dynamic relocation scheme. As shown in Fig. 3, the instruction “A” with compiler address 0x0100 is stored in the physical location with trace address 0x308. Then the other instructions are stored in the data memory in the execution sequence. Furthermore, if necessary, the instruction “A” also can be reassigned to another location in the run time. With this dynamic relocation scheme, the collisions that happen in a one-way set-associative cache are avoided effectively.

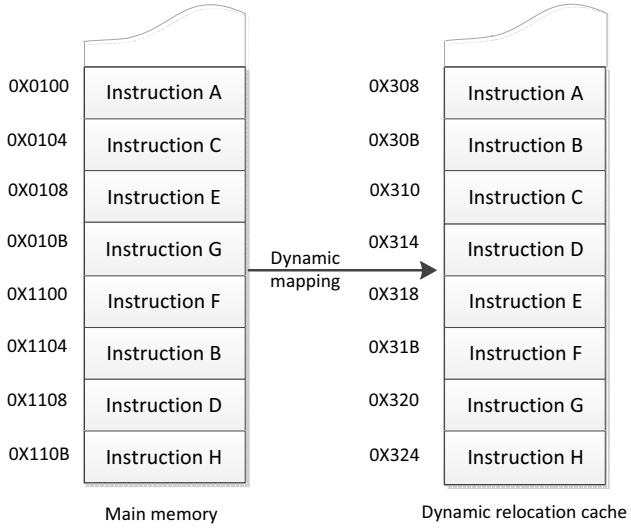

Fig. 3. Dynamic relocation scheme in DR cache

3.3 The Address Mapping Module

The address-mapping module has the following four major functions: (1) mapping the compiler address into the trace address if a corresponding instruction is already stored in the data memory, (2) providing hit signal, (3) assigning a new trace address for an incoming instruction when a cache miss happens, and (4) storing the tag address of the compiler address. As shown in Fig. 4, the requested compiler address launched by the instruction pipeline is divided into two parts: the line address and the tag address. The line address is the low five bits that directly input into the data memory. The remaining bits are input into the address-mapping module as the tag address, and then are mapped into a line number. The trace address is further delivered to the data memory for memory reading in a cache hit or writing in a cache miss.

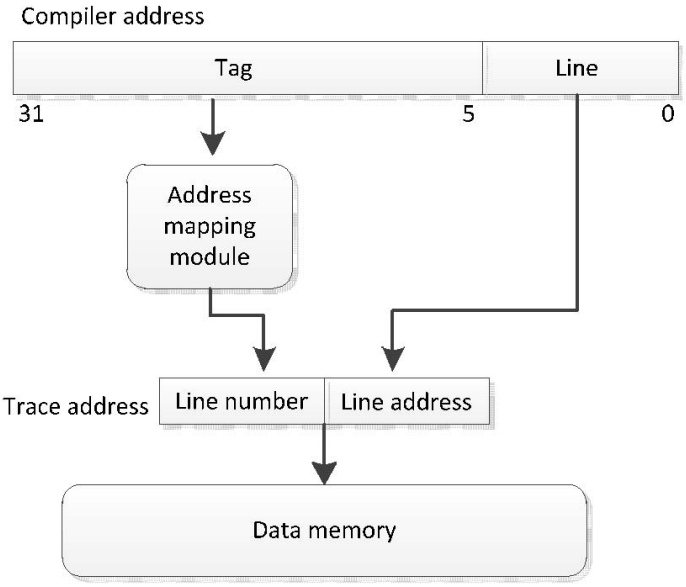

Fig. 4. Request scheme of DR cache

To map the compiler address into the trace address, a set of registers called *location register* are proposed. The location register is used to store the tag address of instructions. Moreover, the tag address is stored in a bit grain. Specifically, two n-bit location registers are used to store one bit of the tag address. When this bit of the tag address is zero, it is stored in the location register 0. On the other hand, when this bit of the tag address is one, it is stored in the location register 1. These two location registers for bit 0 of the tag address are illustrated in Fig. 5. Furthermore, every bit of the location register represents a line of the data memory. Bit zero represents line zero. Bit n represents line n. Therefore, the width of the location register is decided by the size of the data memory.

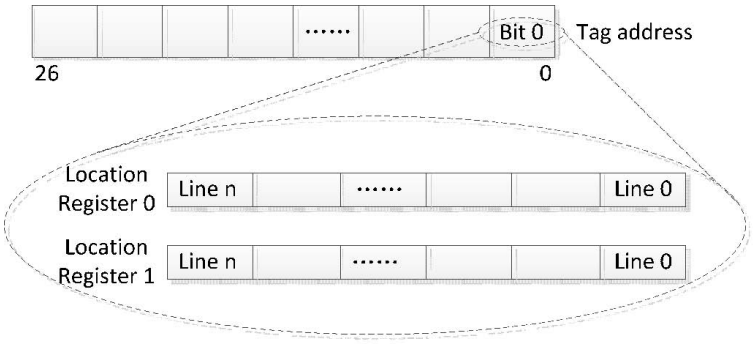

Fig. 5. Location registers

With the location registers, the compiler address can be easily mapped into the trace address. To show

how the location registers work, a simple example with the 4-bit tag address and a 4-line cache memory is illustrated in Fig. 6, where LR30 stores the '0' value of bit 3 within the tag address and LR31 stores the value '1' of bit 3 within the tag address. The instructions with tag addresses "1111", "0000", "1010" and "0101" are stored in line "00", "01", "10" and "11" of the data memory. The tag addresses of these instructions are already stored in the corresponding location registers.

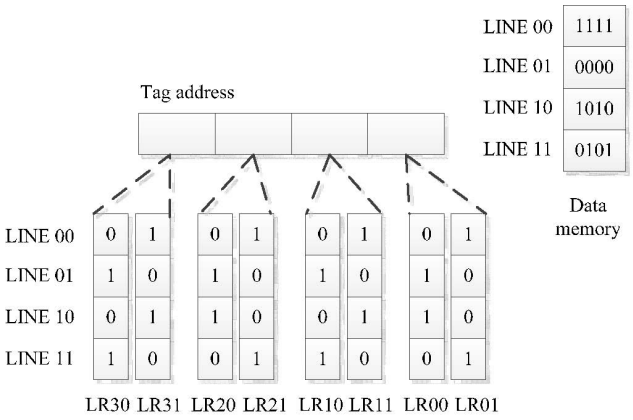

Fig. 6. The location registers example with 4-bit tag address and a 4-line cache memory. The instructions with tag address "1111", "0000", "1010" and "0101" have been stored in line '0', '1', '2' and '3' of the data memory

To map the tag address into the line number, there are two simple steps to do. First, every bit of the requested tag address is used to select one of the two location registers separately. Second, all selected location registers will AND together to get the line number. As shown in Fig. 6, if the requested tag address is "1100", the AND operation of LR31, LR21, LR10 and LR00 will result in "0000", which means there are no instructions with a "1100" tag address that have been stored in the DR cache. If the requested tag address is "1010", the AND operation of LR31, LR20, LR11 and LR00 will result in "0100", which means the instruction with the "1010" tag address has been stored in the line "10" of the DR cache. The line number "10" is further obtained by encoding the "0100".

The microarchitecture of the address-mapping module is illustrated in Fig. 7. There are two processes in the address-mapping module: mapping and updating.

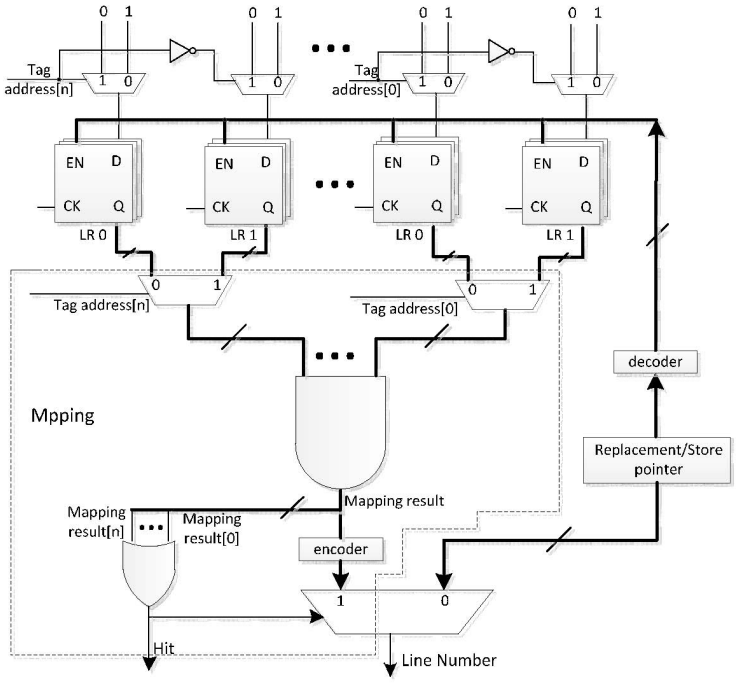

Fig. 7. Microarchitecture of address mapping module

Mapping. For every request from the instruction pipeline, the mapping process is executed to map the requested compiler address into the trace address. This mapping process has been described in the previous paragraphs. The detail hardware architecture is illustrated in Fig. 7. Furthermore, the hit signal is calculated by the OR operation between every bit of the mapping result. The hit signal is zero when a cache miss occurs. The hit signal is one when a cache hit occurs. If the hit signal is positive, the requested instruction will be loaded from the data memory according to the trace address. If the hit signal is negative, the controller will request the main memory with the compiler address, and activate the updating process of the address-mapping module.

Updating. The updating process has two functions: (1) assigning a new trace address for the instruction from main memory and (2) storing the tag address of this instruction. Because the instructions are stored in an execution sequence, we use a store pointer, which is always added one, to assign a new trace address for the incoming instruction. Unfortunately, the overflow happens if the data memory is full. Some old instructions must be replaced. The trace-based storing scheme with overflow situation is illustrated in Fig. 8. Because the store pointer always points to the next storage location, the oldest instruction is always stored at the next location of the newest instruction. Therefore, we let the store pointer to be the replacement pointer. This replacement scheme could be called First-In-First-Replace (FIFR). The FIFR replacement strategy makes sure the instruction is always stored in execution sequence even after the data memory is full, maintaining the trace continuity. Therefore, The FIFR replacement strategy is identical to our trace-based storing scheme. Furthermore, The FIFR replacement strategy is efficient if the data memory is big enough to store a frequently executed block, such as a "for loop". This will be further discussed in section 6.

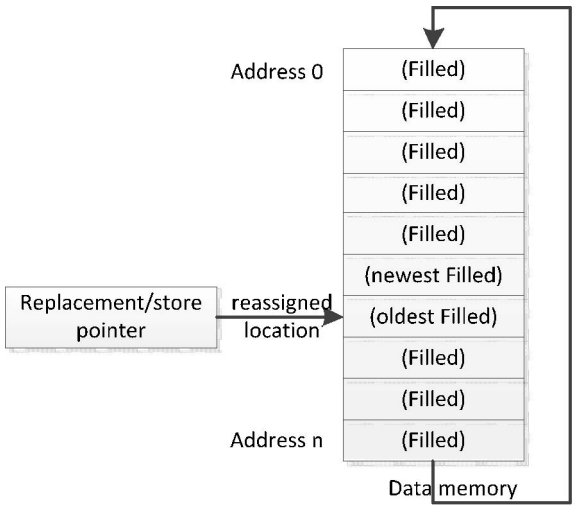

Fig. 8. Trace-based storing scheme with overflow situation

After a new trace address is assigned for the incoming instruction, the corresponding tag address must be stored in location registers. As shown in Fig. 7, the replacement pointer, which is updated according to the FIFR replacement strategy, is decoded to be an enable signal. This enable signal masks most bits of the location registers except the one specified by the replacement pointer. Then, the tag address is stored in the location registers by following rule: the LRx1 will be set to "1" and the LRx0 to "0" if the corresponding bit of tag address is "1", and the LRx1 will be set to "0" and the LRx0 to "1" if the corresponding bit of tag address is "0". Finally, the line number could be the encoded mapping result with positive hit signal or the replacement/store pointer with negative hit signal.

4 Power Consumption Analysis

In this paper, the direct-mapped cache, 2-way set-associative cache and 4-way set-associative cache are introduced for the purpose of comparison. The block diagram of the set-associative cache is illustrated in Fig. 9. With different micro-architecture, the power consumption, runtime performance and area of

caches could vary significantly. To guarantee runtime performance, the baseline of cache microarchitecture in this paper is the one clock loading, which makes sure the instruction pipeline can acquire one instruction from cache in one clock. With this baseline constraint, the tag and data memory in set-associative cache must be accessed in one clock, and the mapping process in the address-mapping module of DR cache must be implemented by pure combinational logic.

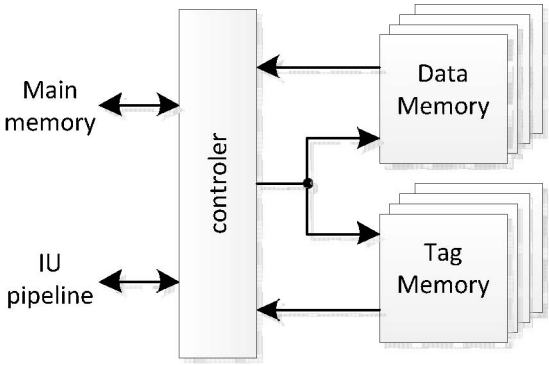

Fig. 9. Microarchitecture of set-associative cache

The sequential accessing, tag first and then data, can reduce the power consumption for both the set-associative cache and the DR cache, eliminating the unnecessary accesses to the data memory. The DR cache can use this scheme without any penalty because the address-mapping module is already sequential with the data memory. Unfortunately, this sequential scheme in set-associative cache must pay the latency penalty, which may be unacceptable in most circumstance because of the above-mentioned baseline constraint. For fairness, we do not adopt this sequential scheme neither for the DR cache nor for the set-associative caches. Furthermore, the tag and data memory of multi-way set-associative cache is implemented as multi-bank memory because of the limitation of the SRAM IP library.

The DR cache is proposed as the power-efficient instruction source. The low-power features of the DR cache are discussed as follow. As shown in Fig. 1, the DR cache uses one single-bank data memory to store instructions, and provides high hit rate by the trace-based storing scheme. On the other hand, the n-way set-associative cache uses n-bank data memory to store instructions. The high hit rate is guaranteed by the multiple locations where one instruction can store. With high hit rate, the power consumption is extremely reduced because of the rarely access to the main memory. Furthermore, the DR cache consumes less power than multi-way set-associative cache because of the single-bank data memory.

The DR cache uses address-mapping module to map the compiler address into the trace address and store this compiler address. The address-mapping module has two processes, the mapping and updating. The mapping process, which consists of some simple “selection” and “AND” operations, is executed for every request. The updating process aiming to update the location registers is executed only when a cache miss happens. Therefore, to optimize the power of the address-mapping module, we use clock-gating technique to freeze the clock of location registers when a cache hit happen. Because the cache hit rate is very high in real applications, the updating process is rarely executed in most circumstances. Moreover, the mapping process consumes few power because the “AND”, “OR” and “selection” operations are very simple. Hence, the clock-gating technique significantly reduces the power consumption of the address-mapping module, which will be confirmed in section 6. On the other hand, the n-way set-associative cache uses the n-bank tag memory to store the mapping information. The address-mapping module with clock gating technique should consume less power than the multi-bank tag memory.

In brief, the DR cache is power-efficient because of the low- power address-mapping module, the high hit rate, and the single-bank data memory.

In brief, the DR cache is power-efficient because of the low- power address-mapping module, the high hit rate, and the single-bank data memory.

5 Evaluation Methodology

We built an RTL (register transfer level) hardware model based on the LEON3 processor to evaluate the DR cache. This hardware model including DR cache or set-associative cache was simulated by using Mentor Modelsim. In the simulation process, some hardware monitors embedded in the cache and instruction pipeline are used to collect the performance data. Finally, Synopsys Design Compiler was used to estimate the power consumption of the caches through its power estimation engine [16]. In this section, we describe the evaluation flow, the benchmark applications and the performance metrics.

5.1 Evaluation Flow

We used a standard ASIC design flow to evaluate the power consumption and runtime performance, which is illustrated in Fig. 10. Firstly, Mentor Modelsim was used to simulate our hardware model through running ten embedded applications. Then the runtime performance including IPC (instruction per clock) and cache hit rate is acquired through the hardware monitors. Since the switch activity of all input ports of the hardware model is necessary for power estimation [16], we dumped the VCD (Value Change Dump) information during the simulation period and used the vcd2saif utility in Synopsys Design Compiler to generate the desired switch activity information.

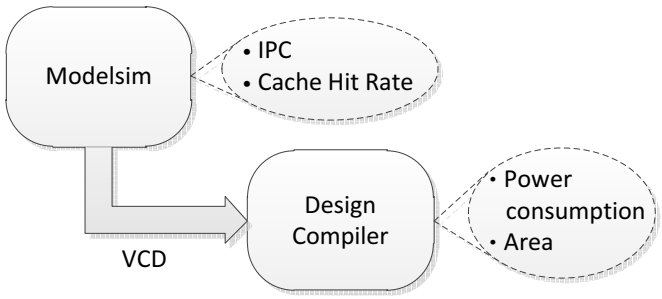

Fig. 10. Evaluation flow for runtime and power performance

We used a commercial 180 nm general technology library under worst-case conditions. To estimate the power consumption of every module, the auto ungroup function of the synthesis EDA tool was shut down. With the switch activity that reflects the switch rate of every signal, Synopsys Design Compiler estimates the internal and dynamic power of caches. Finally, the area of caches was further estimated by this EDA tool.

5.2 Benchmark Programs

Because the simulation based on real hardware model is very slow, we used ten appropriate embedded programs to evaluate the IPC, hit rate, area and power consumption under reasonable time constraint. These programs are listed in Table 1, where the Dhrystone, Whestone and Stanford are standard benchmark program, and the rests are all come from the MiBench program [25]. Furthermore, all evaluation results are averaged across all these benchmark programs.

Table 1. Selected embedded Programs

Programs	Code size (KB)	Category
Dhrystone	58	Standard
Whestone	175	Standard
Stanford	163	Standard
Basicmath	86.9	Automotive
Bitcount	89.3	Automotive
Qsort	110	Automotive
FFT	147	Telecomm
Sha	238	Security
Stringsearch	70.9	Office
Dijkstra_direct	67.1	Network

5.3 Performance Metric

A set of hardware monitors, which record the clock number, instruction number and cache miss number, was used to measure IPC and hit rate. These performance metrics is further calculated by Equations (1) and (2).

$$IPC = \frac{instruction\ number}{clock\ number} \tag{1}$$

$$cache\ hit\ rate = \frac{cache\ hit\ number}{instruction\ number} \tag{2}$$

The instruction number, clock number and cache miss number were measured by three 64-bit counters that were activated by different conditions. The clock counter, which records the elapsed clock cycles, was activated by system clock at all time. The instruction counter was activated by system clock only when the pipeline moved forward normally. For convenience and accuracy, the cache miss counter was used, and was activated by system clock only when the main memory feedback a notification. Then we use the instruction number minus the cache miss number to get the cache hit number.

The power consumption of caches was estimated by Synopsys Design Compiler. In addition, the power consumption of the main memory was estimated by Eq. (3), where the "IPC x 10^8" represents the executed instructions in one second (the frequency is 100 MHZ). We assume the main memory was dedicated to instructions, and thus was activated only when a cache miss happens. We assume the main memory is the 64 MB Samsung K4X51163PC SDRAM [26]. The refresh power is ignored for simplification. The read energy E_{read} is 11.747 nJ according to the [26]. Finally, the total power consumption of the memory subsystem is estimated by Eq. (4), where the P_{cache} is estimated by the power estimation engine of Synopsys Design Compiler.

$$P_{main} = E_{read} \times IPC \times 10^8 \times cache\ miss\ rate \tag{3}$$

$$P_{total} = P_{cache} + P_{main} \tag{4}$$

6 Evaluation Result and Discussion

We compare the DR cache with the direct-mapped cache, 2-way set-associative cache and 4-way set-associative cache in runtime performance and power consumption including the power with same cache size, the smallest power, and the power with comparable area. Finally, the reason of the power saving is discussed.

6.1 IPC and Hit Rate

The runtime performance is evaluated in two aspects: the average hit rate and IPC. The hit rate reflects the performance of a cache system. On the other hand, the IPC, which could be influenced by many factors including cache architecture, load/store blocking, and exception, reflects the performance of a processor. The computational method of these performance metrics is described in Equations (1) and (2). Because the clock and instruction numbers of different programs have large variations, their arithmetic mean is not proper. Instead, we use Equations (5) and (6) to calculate the average IPC and cache hit rate of the ten programs. The 2-way and 4-way set-associative cache use LRU replacement strategy. The average hit rate and IPC is illustrated in Fig. 11 and Fig. 12.

$$IPC_{average} = \frac{\sum_{i=1}^{10} (instruction\ number)_i}{\sum_{i=1}^{10} (clock\ number)_i} \tag{5}$$

$$(cache\ hit\ rate)_{aver.} = \frac{\sum_{i=1}^{10}(icache\ hit\ num.)_i}{\sum_{i=1}^{10}(instruction\ num.)_i} \quad (6)$$

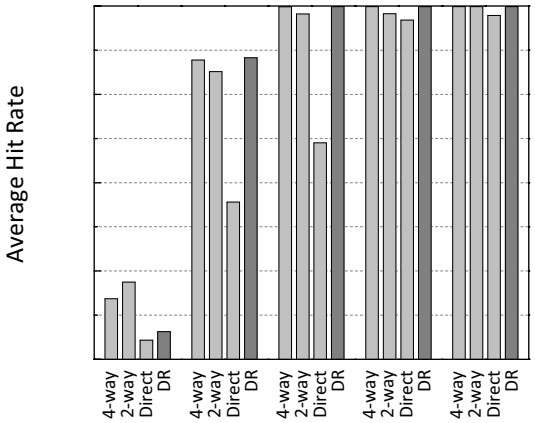

Fig. 11. Average cache hit rate

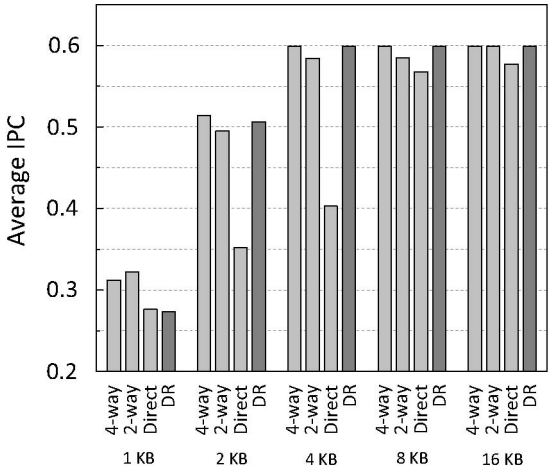

Fig. 12. Average IPC

As shown in Fig. 11, when the cache size is 1-kB, the DR cache has lower cache hit rate than the set-associative cache and higher cache hit rate than the direct-mapped cache. The reason of this reduction of hit rate is discussed as follow. In the DR cache, the FIFR replacement strategy replaces the oldest instruction, which makes the data memory looks like a circular ring. An instruction will be replaced if the store pointer travels back to it again. This replacement strategy is designed with the consideration of hardware simplicity and trace continuity. However, if the data memory is too small to hold a frequently executed block, such as a "for loop", the oldest instructions that will be replaced could be the one that will be frequently used in the near future. Hence, a small data memory in the DR cache could cause the inefficient replacement, which decreases the cache hit rate. However, the multi-way set-associative cache avoids this inefficient replacement by the LRU replacement strategy.

When the cache size is higher than 2-KB, the DR cache has higher cache hit rate than other caches. Therefore, in our application environment, we infer that the 1-KB data memory is not enough to store the frequently executed block, but the 2-KB data memory is. With the increasing of the cache size, the hit rate finally approaches to some value that indicates the limitation of the cache architecture. The DR cache

has almost same hit rate with the 4-way set-associative cache. With the same single-bank data memory, the DR cache has higher cache hit rate than the direct-mapped cache.

Furthermore, the cache hit rate is sensitive to the application environment. As shown in Fig. 11, the cache hit rate of the 4-way set-associative cache is comparable to that of the 2-way set-associative cache. This means that the 2-way set-associative cache is enough to deal with the collisions (illustrated in Fig. 2) in our application environment.

As shown in Fig. 12, the trends for the average IPC is similar to those for the hit rate. In general, higher hit rate leads to higher IPC because the cache access that consumes one clock is much faster than the main memory access that consumes many clocks. However, the DR cache has lower IPC than the 4-way set-associative cache when the cache size is 2-KB. The reason of this reduction of IPC is discussed as follows. In the DR cache, the cache line is managed as a unity. All contents of the line are update together at a time when a miss happens. Hence, some unused instructions, which are located before the requiring one, are fetched into the cache. This consumes extra clocks. Furthermore, the pre-fetch technique is also available in the set-associative caches, but only the instructions that are located after the requiring one are loaded. Superficially, the pre-fetch technique is good for increasing cache hit rate. However, from the evaluation results, we can infer that fetching the instructions, located before the requiring instruction, is inefficient in our test environment.

With the increasing of the cache size, the hit rate approaches to one, and the IPC approaches to the maximum value that is decided by the cache architecture. Therefore, increasing the cache size is a straightforward method to improve the runtime performance if we do not consider the energy consumption.

In brief, when the cache size is higher than 2-KB, our DR cache has almost the same runtime performance as the 4-way set-associative cache.

6.2 Area and Power Consumption

The area with gate-level unit is illustrated in Fig. 13. As shown in Fig. 13, the DR cache occupies less area than the 4-way set-associative cache when the cache size is 1-kB. However, with the increasing of the cache size, the area of the DR cache increases much faster than that of other caches. Specifically, when the cache size is 16-KB, the DR cache occupies 2.6 times area compared to the 4-way set-associative cache. The reason of this high-speed increasing of area is described as follows. In the address-mapping module of the DR cache, the location registers, which is used to store the tag address, is implemented by flip-flops. Furthermore, two registers are used to store one bit of the tag address. On the other hand, the tag memory in the set-associative cache is normally implemented by SRAM. From the density point of view, SRAM is better than register. Therefore, the area increasing of the DR cache is faster than that of the set-associative caches.

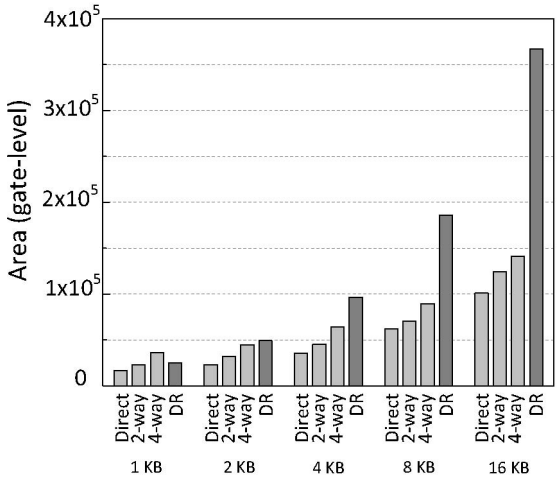

Fig. 13. Area with gate-level unit

The normalized power consumption is shown in Fig. 14, where the 4-way set-associative cache is the

reference case. As shown in Fig. 14, the DR cache has the smallest power in all cache size configurations. Specifically, when the cache size is 4-KB, the DR cache has 83%, 69% and 64% reduction in power consumption compared to the 4-way set-associative cache, the 2-way set-associative cache and the direct-mapped cache.

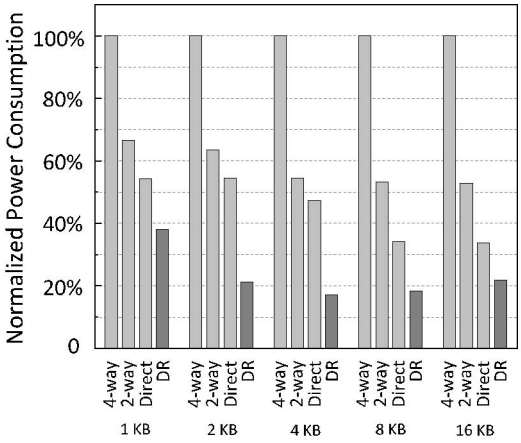

Fig. 14. Normalized power consumption

The 4-way set-associative cache has larger power than the 2-way set-associative and direct-mapped cache, although they have same cache size. The reason of this is described as follow. As described in [1], the energy consumption of SRAM memory has a litter variation when the memory size changes because of its inherent circuit structure. For example, the access energy of 0.5-KB memory is 0.196 nJ, and the access energy of 2-KB memory is only 0.203 nJ. If the 4-way set-associative, 2-way set-associative and direct-mapped caches have same 2-KB cache size, the access to the data memory of the 4-way set-associative cache consumes maximum power because the data memory is consisted of four 0.5-KB memories. Similarly, the direct-mapped cache consumes minimum power because its data memory is consisted of one 2-KB memory.

We show the total power consumption of the direct-mapped cache, the 2-way set-associative cache, the 4-way set-associative cache, and the DR cache in Fig. 15. The power curves have the parabola-liked shape. The reason of this is described as follow. Firstly, with the increasing of the cache size, the cache hit rate increases and finally approached to one. Hence, the reduction in the access power to the off-chip main memory contributes to the reduction part of the power curve. Secondly, with the increasing of the cache size, the power consumption of the on-chip cache increases slowly. This contributes to the increasing part of the power curve. Hence, for a specific application environment, there is a best suitable cache size for a cache in power consumption.

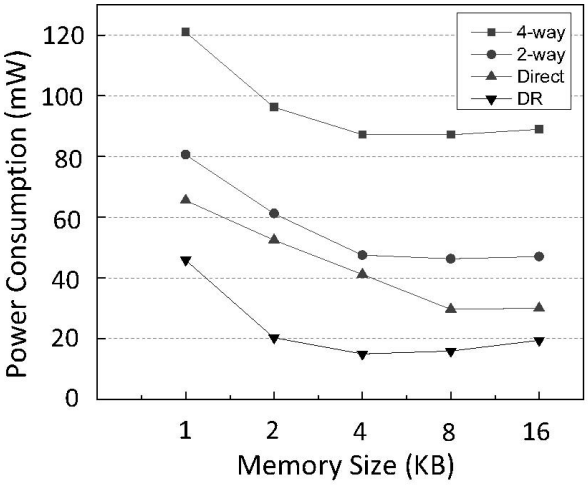

Fig. 15. Total power consumption when the cache size increases from 1 KB to 16 KB

As shown in Fig. 15, the power curve of caches has a lowest point. We compare the lowest limits of power consumption, and thus find out the best cache architecture in power consumption. According to Fig 15, we choose the 8-KB direct-mapped cache, the 8-KB 2-way set-associative cache, the 8-KB 4-way set-associative cache, and the 4-KB DR cache for this comparison. The smallest power is shown in Fig. 16, where we divide the power into two parts, the on-chip cache and the off-chip main memory. As shown in Fig. 16, the DR cache consumes less power than other caches, which implied that the DR cache is more efficient than other caches in power consumption. Furthermore, when caches reach their lowest limits of power consumption, the power consumption of the SDRAM memory is very small. Hence, to optimize the power of the cache system, choosing a proper cache size is very important.

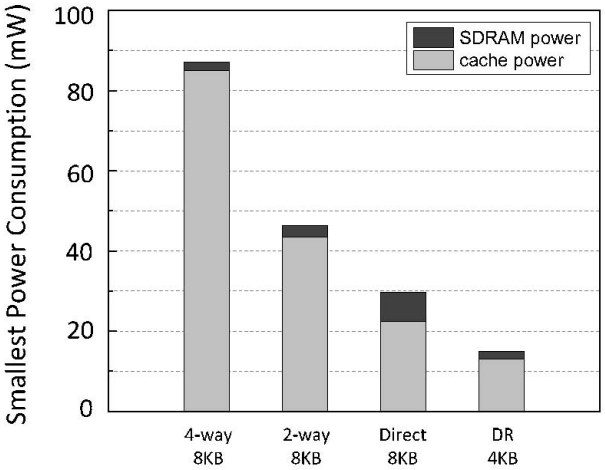

Fig. 16. Comparison of lowest limits of power consumption

Area is normally considered as the secondary design object since it is always sufficient under the advanced semiconductor technology. However, in some special condition, the area becomes important and needs to be carefully considered. Hence, we compare the four caches with comparable area. According to Fig. 13, we choose the 8-KB direct-mapped cache, the 8-KB 2-way set-associative cache, the 4-KB 4-way set-associative cache, and the 2-KB DR cache for this comparison. The power consumption with comparable area is shown in Fig. 17. Furthermore, the normalized area of these four caches is shown in Fig. 18, where we choose the smaller DR cache to guarantee the fairness. Again, our DR cache consumes less power than other caches. As mentioned before, when the cache size is higher than 2-KB, the DR cache has almost same runtime performance with the 4-way set-associative cache. Furthermore, with comparable area, the DR cache has approximate 83% less power consumption compared to the 4-way set-associative cache.

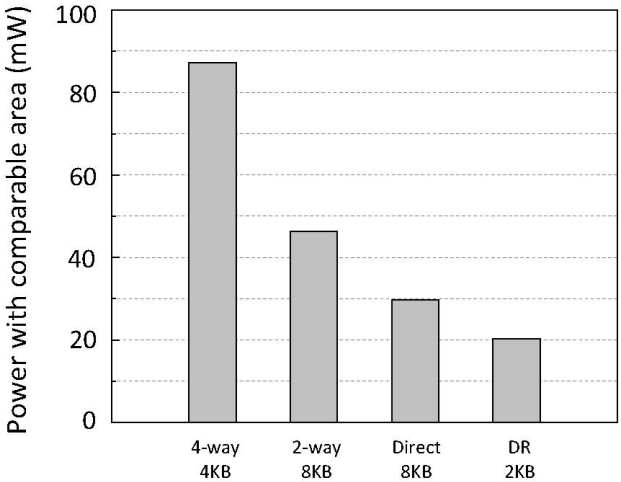

Fig. 17. Comparison of power consumption of all cashes with comparable areas

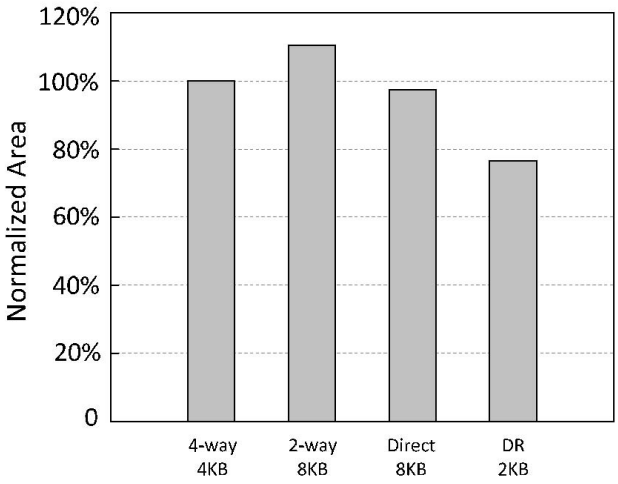

Fig. 18. Normalized area of four specific caches that have comparable area

6.3 The Source of Power Saving

The DR cache with single-bank data memory and address mapping module is power-efficient. With 4-KB cache size, it provides same performance in hit rate and 83% improvement in power consumption compared to the 4-way set-associative cache. The source of this power saving is discussed in this section.

The dynamic address mapping is a key technique for the trace-based storing scheme. As mentioned in Section 4, we use the clock-gating technique to optimize the power of the address-mapping module. To estimate the effect of the clock gating, an original design without clock gating is implemented. The power consumption of these two designs is illustrated in Fig. 19. Without the clock gating, the power consumption of the original address-mapping module becomes very large even when the cache size is only 2-KB. However, with the clock gating, the power consumption of the address-mapping module is dramatically reduced. Therefore, the clock gating makes the dynamic address-mapping scheme practical.
We compare the address-mapping module of the DR cache with the tag memory of the set-associative caches in power consumption. The evaluation results are illustrated in Fig. 20, where they are normalized by making the 4-way set-associative cache as the reference case. As shown in Fig. 20, the address-mapping module has a 99% and 96% less power consumption compared to the 4-way set-associative with 1-KB and 16-KB cache size. Therefore, the total power consumption of the DR cache is mostly contributed by the single-bank data memory.

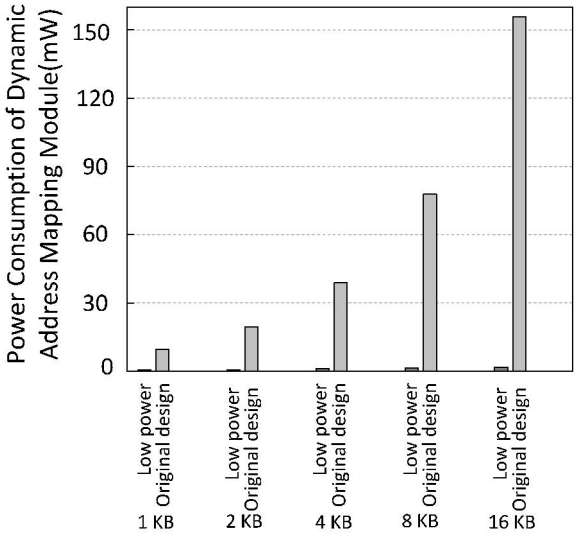

Fig. 19. Power consumption of the address-mapping module, comparing the original and low power design

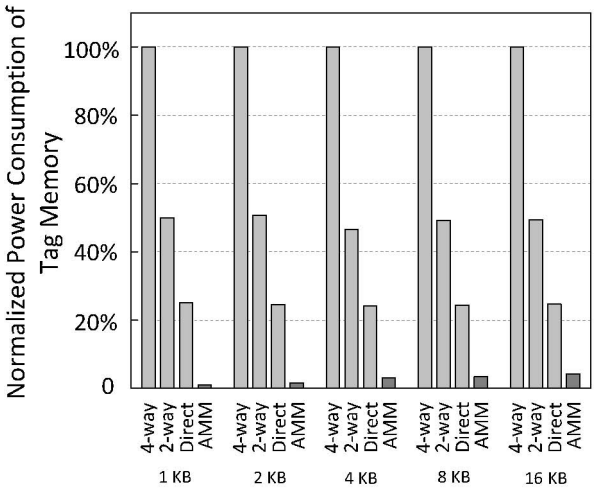

Fig. 20. Normalized power consumption for the tag memory in traditional cache and the address-mapping module in DR cache. (AMM: Address-mapping Module)

The power consumption of memory subsystem is primarily contributed by the main memory and the on-chip cache with controller, data memory, and tag memory (address mapping module). The power breakdown of the memory subsystem with 4-kB cache is illustrated in Fig. 21. It is clear that the power saving of the DR cache is mostly come from three aspects: (1) the high hit rate, (2) the low-power address-mapping module, and (2) the single-bank data memory. The power consumption of the SDRAM memory in the memory subsystem with DR cache is lowest because of its highest cache hit rate. The address-mapping module consumes much less power than other tag memories because of the efficient clock-gating technique. Moreover, the single-bank data memory in the DR cache consumes less power than the multi-bank data memory in the multi-way set-associative cache, because of the circuit structure of SRAM.

In brief, the DR cache is power-efficient because of the high hit rate, the low-power address-mapping module and the single-bank data memory. This have been discussed in Section 4, and further verified in this section.

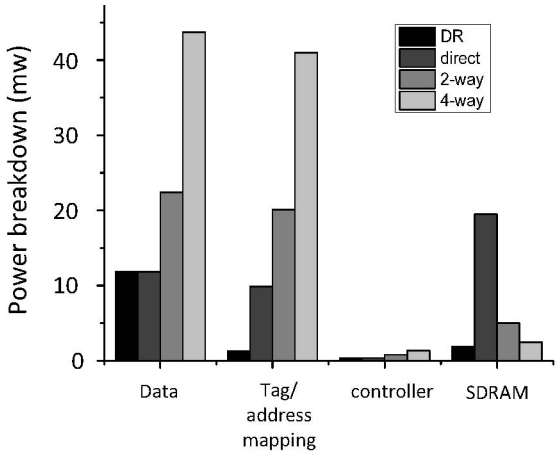

Fig. 21. Power breakdown

7 Conclusion

In this paper, the dynamic relocation cache is presented for low-power processor to replace the set-associative instruction cache. The DR cache with the features of low power and high hit rate uses a trace-based scheme to store the instructions. This trace-based scheme assigns a new trace address to the incoming instruction according to the execution sequence. For implementing this trace-based storing

scheme, the address- mapping module is used to store the compiler address and map the compiler address to the trace address. To deal with the overflow situation of the data memory, the First-In-First-Replace strategy is used to guarantee that the instruction is always stored in an execution sequence even after the data memory is full, maintaining the trace continuity. Furthermore, a clock-gating technique is used to improve the power of the address-mapping module.

We evaluate the proposed DR cache through a RTL hardware model. To insure the accuracy, a standard ASIC design flow is used to evaluate the power consumption and runtime performance. The evaluation results show that, the 4-KB DR cache has 83% less power consumption and same hit rate compared to the 4-KB 4-way set-associative cache. Furthermore, the DR cache also surpasses other caches in the power with comparable area and the smallest power.

The DR cache with the advantages of high hit rate and low power is very suitable for embedded processors, because the cache size of embedded processors is normally small. The DR cache also could be used in desktop processors or multiprocessors, if the area cost is tolerable or a small DR cache can hold the frequently executed block in their application environment. However, the DR cache, which is proposed for instruction, may not be suitable for data, because the miss rate of data cache is normally high. Higher miss rate causes more frequent updating of the location registers in the address-mapping module. This causes the increasing of power consumption.

The multi-way set-associative cache has been widely adopted because of its good runtime performance. However, if the power consumption is also considered, the multi-way set-associative cache may not be a good choice. The DR cache has the features of high hit rate and low power. Hence, it could be a preferable choice in low-power processors under some area constraints.

Acknowledgements

The authors would also like to thank their laboratory team member's assistance. This work was supported by the financial aid of the National Science-technology Support Plan Projects "Development and application demonstration of temperature monitoring networking platform of third party pharmaceutical cold chain logistics" under grant No. 2014BAH23F05, and the National Natural Science Foundation of China under grant No. 61274133.

References

[1] B. Egger, S. Kim, C. Jang, J. Lee, S. L. Min, H. Shin, Scratchpad memory management techniques for code in embedded systems without an MMU, IEEE Trans. Computers 59(8)(2010) 1047-1062.

[2] L. Bloom, M. Cohen, S. Porter, Considerations in the design of a computer with high logic-to-memory speed ration, in: Proc. Gigacycle Computing Systems, 1962.

[3] J. S. Liptay, Structural aspects of the system/360 model 85, part II: The Cache, IBM Systems Journal 7(1)(1968) 15-21.

[4] ARM, The ARM Cortex-A9 processors. <http://www.arm.com>, 2009.

[5] A. J. Smith, Cache memories, ACM Computing Surveys 14(3)(1982) 473-530.

[6] J. Kin, M. Gupta, Filtering memory references to increase energy efficiency, IEEE Trans. Computers 49(1)(2000) 1-15.

[7] A. Janapsatya, S. Parameswaran, A. Ignjatovic, HitME: Low power hit memory buffer for embedded systems, in: Proc. 2009 Asia and South Pacific Design Automation Conference, 2009.

[8] C. Yang, A. Orailoglu, Power-efficient instruction delivery through trace reuse, in: Proc. 15th International Conference, 2006.

[9] E. Rotenberg, S. Bennett, J.E. Smith, A trace cache microarchitecture and evaluation, IEEE Trans. Computers 48(2)(1999) 111-120.

[10] Y.-Y. Tsai, C.-H. Chen, Energy-efficient trace reuse cache for embedded processors, IEEE Trans. Very Large Scale

Integration (VLSI) Systems 19(9)(2011) 1681-1694.

[11] O. Avissar, R. Barua, An optimal memory allocation scheme for scratchpad-based embedded systems, IEEE Trans. Embedded Computing Systems 1(1)(2002) 6-26.

[12] B. Egger, J. Lee, H. Shin, Dynamic scratchpad memory management for code in portable systems with an MMU, ACM Trans. Embedded Computing Systems 7(2)(2008) 1-38.

[13] H.-Y. Luo, S.-J. Wei, D.-H. Guo, The dynamic relocation cache and its energy consumption model for low power processor, in: Proc. 2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification (ASID), 2011.

[14] A. Agarwal, J. Hennessy, M. Horowitz, An analytical cache model, ACM Trans. Computer Systems 7(2)(1989) 184-215.

[15] Aeroflex Gaisler, GRLIB IP core user's manual, <http://www. gaisler.com>. 2010.

[16] Synopsys, Design compiler user guide. <http://www.syno- psys.com>, 2008.

[17] S. Steinke, N. Grunwald, L. Wehmeyer, R. Banakar, M. Balakrishnan, P. Marwedel, Reducing energy consumption by dynamic copying of instructions onto onchip memory, in: Proc. 15th Int'l Symp. System Synthesis (ISSS '02), 2002.

[18] B. Egger, J. Lee, H. Shin, Dynamic scratchpad memory management for code in portable systems with an MMU, ACM Trans. Embedded Computing Systems 7(2)(2008) 1-38.

[19] J. Kin, M. Gupta, W.H. Magione-Simth, Filter cache: an energy efficient memory structure, in: Proc. 30th Int. Symp. Microarch., 1997.

[20] N. Bellas, I. Hajj, C. Polychronopoulos, G. Stamoulis, Energy and performance improvements in microprocessor design using a loop cache, in: Proc. Int. Conf. Comput. Des., 1999.

[21] W. Tang, R. Gupta, A. Nicolau, Design of a predictive filter cache for energy savings in high performance processor architectures, in: Proc. Int. Conf. Comput. Des., 2001.

[22] A. Sodani, G.S. Sohi, Dynamic instruction reuse, ACM SIGARCH Computer Architecture News 25(2)(1997) 194-205.

[23] D. Charles, A. R. Hurson, N. Vijaykrishnan, Improving ILP with instruction-reuse cache hierarchy, in: Proc. 5th Int. Conf. Algorithms Arch. for Parallel Process, 2002.

[24] J.S. Hu, N. Vijaykrishnan, M.J. Irwin, M. Kandemir, Using dynamic branch behavior for power-efficient instruction fetch, in: Proc. IEEE Comput. Soc. Annu. Symp. VLSI, 2003.

[25] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, R.B. Brown, MiBench: A free, commercially representative embedded benchmark suite, in: Proc. IEEE 4th Annual Workshop Workload Characterization, 2001.

[26] Samsung Semiconductor, K4X51163PC Mobile DDR SDRAM. <http://www.samsung.com/products/semiconductor /MobileSDRAM/>, 2005.