國立交通大學

電子工程學系 電子研究所

碩 士 論 文

採用循序路存取之
低功率集合關聯快取記憶體架構

Sequential Way-Access Set-Associative Cache Architecture for Low Power

研 究 生：丁之暉

指導教授：黃俊達 博士

中 華 民 國 九 十 六 年 九 月

採用循序路存取之
低功率集合關聯快取記憶體架構
Sequential Way-Access Set-Associative Cache Architecture for Low Power

研　究　生：丁之暉　　　　　　Student: Chih-Hui Ting
指導教授：黃俊達 博士　　　　Advisor: Dr. Juinn-Dar Huang

國立交通大學

電子工程學系　電子研究所

碩士論文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical & Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering & Institute of Electronics

September 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年九月

採用循序路存取之
低功率集合關聯快取記憶體架構

研究生：丁之暉　　　　　指導教授：黃俊達　博士

國立交通大學

電子工程學系 電子研究所碩士班

摘　　要

近代快取記憶體架構藉由使用集合關聯快取記憶體來減少快取記憶體的失誤率同時並保有快速的存取速度。然而集合關聯快取記憶體卻造成可觀的功率消耗。其原因在於每次快取記憶體存取皆會觸發所有的路，然而實際上只有命中的路是有意義的。為了減少快取記憶體的功率消耗，一個可行的方法就是減少每次快取記憶體存取時被觸發的記憶體陣列數目。

在本篇論文中，我們檢驗藉由循序路存取的方式來減少每次快取記憶體存取時被觸發的路數目。循序路存取在每次快取記憶體存取時依序搜尋每一路，從第一路到最後一路，當某一路命中時，則搜尋的動作結束。當快取記憶體命中在一個較早搜尋的路時，較晚搜尋的路的存取就可以被排除。藉由較精明的區塊放置及取代策略，我們可以增加最早被搜尋的路的命中率因而進一步減少每次快取記憶體存取時被觸發的路的數目。由於每個週期中被存取的路是預先確定的，因此在傳統集合關聯快取記憶體中命中信號用於選擇命中區塊的多工器所造成的負荷可以被排除。這些減少的命中信號負荷可降低快取記憶體的存取週期因此能夠抵消增加的存取周期數。實驗顯示，相對於傳統的32KB二路集合關聯快取記體，同樣大小之採用循序路存取二路集合關聯記憶體平均能減少23.8%的功率-延遲乘積。

Sequential Way-Access Set-Associative Cache Architecture for Low Power

Student: Chih-Hui Ting   Advisor: Dr. Juinn-Dar Huang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

Abstract

Modern cache architectures utilize set-associative notion to increase the hit rate while maintaining fast access time. However, set-associative cache results in tremendous waste in power since all the ways are probed on each cache access but actually only the matching way is required. In order to reduce cache power consumption, a possible solution is to reduce the number of activated memory arrays on each cache access.

In this thesis, we examine the concept of sequential way access to reduce the number of ways being activated on each cache access. Sequential way access probes each way sequentially, from the first way to the last way on each cache access. The access sequence terminates when a cache hit is detected. A cache hit in an earlier accessed way can eliminate subsequent accesses on later ways. By using certain smarter placement and replacement policies, the hit rate on the earliest accessed way can be increased therefore the number ways being activated on each cache access can be further reduced. Since the accessed way on each cycle is predetermined, the fanout on the hit-signal to the multiplexer which selects the matching data block in conventional set-associative cache can be eliminated. The reduced loading on the hit-signal reduces cache cycle time and therefore can compromise the increased cache access cycle count. Experimental result shows that a 32KB 2-way sequential way-access set-associative cache reduces the power-delay product by an average of 23.8% compared to the conventional set-associative cache of the same size and associativity.

誌　謝

首先, 我要感謝我的指導教授–黃俊達博士, 在碩士兩年當中給我的支持和鼓勵, 讓我能有良好的研究環境, 在自由的學習風氣之下, 培養出獨立研究的能力, 又能隨時給予寶貴的意見及指導, 對老師的感激之情, 並非以簡短的文句可以表達。

當然也要感謝養育我長大的父母對我的栽培, 沒有他們, 就沒有今日的我。接著要感謝的是所有來參與口試的所有教授們, 黃婷婷教授, 和林永隆教授, 百忙當中抽空前來指導我, 讓我受益匪淺, 也讓我得到了寶貴的經驗, 謝謝你們。

最後也要感謝所有實驗室的同仁們, 許哲霖、李南興、張詠翔學長以及學弟們, 跟大家一起修課、做實驗、和討論及分析研究成果更是我人生旅途中一段最值得珍惜的回憶, 希望未來在讀書或工作還有機會一起努力。

希望這篇論文能對人類社會有小小的貢獻, 如此一來在辛苦也就值得了, 再次謝謝大家的幫忙。

Contents

List of Tables

List of Figures

Chapter 1 Introduction

1.1. Motivation

Continuous advances in VLSI technology have led to more powerful and sophisticated microprocessor designs. The significant improvements in performance are due to both better circuit design and fabrication technology. On the other hand, state-of-the-art designs also cause power consumption of microprocessor to increase dramatically in recent years. The power density of current chips is already higher than a hot plate. If this trend holds, the power density of chips will close to a nuclear reactor before the year 2010 [1].

High-performance cache dissipates significant power due to charging and discharging of highly capacitive bit lines and sense amplifiers. Several published reports have also shown that cache consumes substantial fraction of overall chip power. The on-chip caches of StrongARM SA110 dissipate about 43% of the total chip power [2]. In the 300 MHz bipolar CPU reported by Jouppi et al [3], 50% of power is dissipated by caches.

To increase cache efficiency, modern microprocessors employ set-associative cache, which increases the performance by reducing thrashings among cache blocks that map to the same cache set and therefore reduces conflict miss. To achieve fast access, set-associative cache probes both tag and data arrays in parallel, and then selects the data block form the matching way according to tag comparison result. The advantage of parallel access is that no additional memory access is required after the matching way is found. But on the other hand, parallel access causes tremendous waste in power since all the ways are probed on each cache access but actually only the matching way is required.

One possible solution for this problem is to access tag array and data array serially, which is used by Alpha21164's L2 cache [4]. The concept was first introduced by Hasegawa et al [5] and was referred to as *phase cache* by [6]. In phase cache, tag arrays are probed first to determine the matching way. Once the matching way is determined, only the data array of the matching way is *then* accessed. This reduces the number of data arrays being accessed and thus reduces cache power, especially when the associativity is high. But on the other hand, as opposed to parallel tag-data access in conventional set-associative cache, serial tag-data access has to wait until the matching way is known before access on data array can take place and therefore is not quit suitable for time critical L1 cache. While serial tag-data access suffers from long latency, the concept of using serial access to reduce the number of arrays being probed does provide means to reduce cache power consumption.

1.2. Proposed Architecture

In order to achieve cache low power consumption while maintaining fast access clock rate, we propose a cache architecture call *sequential way-access set-associative cache*. Sequential way-access set-associative cache accesses each way (both tag and data array) sequentially, and eliminates subsequent accesses when a cache hit is found. Unlike Way Predicting Cache [6][7], which requires a prediction mechanism to determine which way should be probed first, Sequential way-access set-associative cache ranks a priority of each way and always probes the way of the highest priority first. Since tag and data array are probed in parallel for each way access, the access latency for each way is about the same as conventional set-associative cache. If the first way hits, subsequent accesses on other ways are saved with the access latency almost identical to conventional set-associative cache. On the other hand, additional access cycles and memory accesses are required if the first way misses. By using more efficient placement and replacement techniques, we are able to increase the hit rate on the earliest accessed way by placing recently accessed blocks in the earliest accessed way. This further reduces the number of redundant memory accesses and eases additional access latency. Experimental result shows that the proposed architecture has a lower power-delay product compared to the conventional set-associative cache of the same size and associativity, especially when the miss penalty is high. A 32KB 2-way *sequential way-access set-associative cache* reduces the power-delay product by an average of 23.8% compared to the conventional set-associative cache of the same size and associativity

.

1.3. Thesis Organization

The remainder of the thesis is organized as follows: Chapter 2 discusses several related works; Chapter 3 gives a detailed description on the proposed model; Chapter 4 describes the experiment setup and simulation result; and Chapter 5 concludes the thesis.

Chapter 2 Related Works

2.1 Way-Predicting Set-Associative Cache

Several approaches have been proposed to reduce cache power consumption by reducing the number of access on memory arrays [5, 6, 7, 8, 13, 14]. One such approach is Way-Prediction Set-associative cache [7]. Way-Predicting Set-Associative Cache speculatively chooses one way to probe prior to the actual cache access base on certain prediction mechanism. If the prediction is correct, the cache access completes in a single cycle as shown in Figure 3.1(a). Since only the predicted way is activated, cache power consumption is reduced compared to conventional set-associative cache. On the other hand, if the predicted way misses, as shown in Figure 3.1(b), additional access cycle is required to probe the remaining ways in order to search for the matching way which increases the cache access cycle count. To make things worse, since all the remaining ways are activated, no memory access is saved compared to conventional set-associative cache.

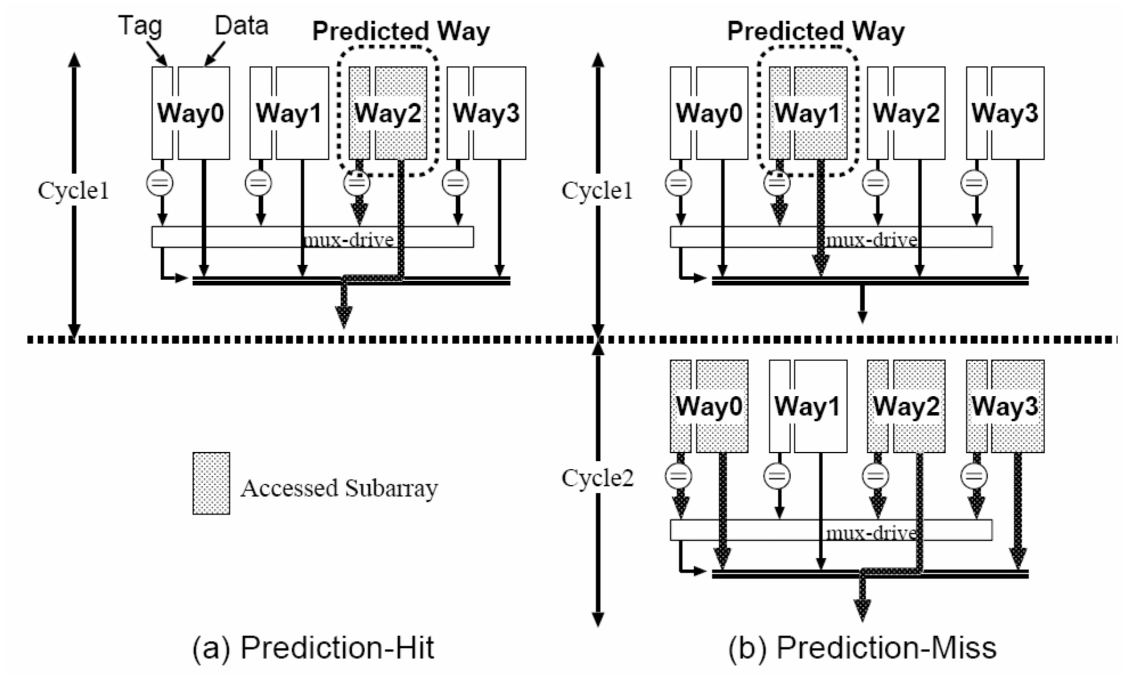


Figure 2.1: Way-Predicting Set-Associative Cache

A correct prediction reduces cache energy consumption while maintaining high access speed (in terms of access cycle count). On the other hand, an incorrect prediction not only fails to reduce cache energy but increases cache access cycle count. Consequently prediction accuracy plays an important role on cache energy-delay product. The average energy consumption (E_{cache}) and the average cache hit cycle count (T_{cache}) for each cache access on the way-predicting 4-way set-associative cache can be expressed as follows:

$$E_{cache} = (E_{Tag} + E_{Data}) + (1 - \mathrm{PHR}) \times (3\,E_{Tag} + 3\,E_{Data}) \qquad (1)$$

$$T_{cache} = 1 + (1 - PHR) \times 1 \qquad (2)$$

- E_{Tag}, E_{Data}： Energy consumed for accessing a tag-array and data array, respectively.
- PHR： Prediction hit rate.

To increase accuracy of way prediction, Way-Predicting Set-Associative cache employs a way-prediction policy based on an MRU (Most Recently Used) algorithm. This policy requires a flag to store the MRU information for each set. All flags are stored in an MRU way-prediction table which is accessed using the same address that determines the activated set. In the case of a 16KB 4-way set-associative cache, the prediction flag is 2 bits and therefore additional 4KB of memory storage is required for the MRU way-prediction table. The predicted way number is read out before the actual cache access takes place in order to get the prediction.

Although Way-Predicting Set-Associative cache seems attractive in reducing cache power, there are several drawbacks. Its way-prediction policy requires a large way-prediction table to store the MRU information for each set. Looking up such a large table also causes substantial power consumption. Furthermore, since actual cache access can not take place until the predicted way is read out from the way-prediction table and since way-prediction table lookup requires the actual data address as its index, the prediction procedure cannot be hidden and therefore cause additional cache access time even if the prediction is correct.

To reduce prediction time, using different information other than the actual data address as index for the way-prediction table allows prediction process to start before the data address is calculated and can therefore hide the prediction time to early stage. Two possible sources for index are: the load PC address and the approximate data address formed by XORing the load's source register with the load offset [8]. The load PC is available much earlier than the approximate data address but the approximate data address is more accurate since a single load instruction may reference data with widely varying access patterns [9].

To obtain a better balance between energy consumption and overall speed for way-predicting cache, two different access schemes are proposed by Huang, et al [10] based on the concept of phased cache.

In the first scheme, called *Fall Back Phased*, the first probe activates only the predicted way in the same way as Way-Predicting Set-Associative cache. If the first probe misses, the second probe acts as a phased cache which probes all the remaining tag arrays but not the data arrays. If a match is detected, the corresponding data array is then accessed. Otherwise cache miss encounters and no data arrays are then accessed. This scheme favors energy saving at the expense of speed.

On the other hand, the second scheme, called *predictive phased*, activates all tag arrays and data arrays of the predicted way in the first probe. In this case, additional tag arrays are activated even if the prediction is correct. However, if the prediction is incorrect, the matching way (if any) is known and is subsequently accessed. This scheme has the same access speed as Way-Predicting Set-Associative cache but might not be as energy-efficient as Way-Predicting Set-Associative cache, since more tag arrays are activated on prediction hit but less data arrays are activated on prediction miss. This scheme favors when the prediction accuracy is low.

2.2 Non-Uniform Cache Architecture (NUCA)

Continuous advances in VLSI technology increase the processor clock rate dramatically in recent years. On the other hand, improvements in memory speed have not kept pace with processor speed and result in ever widen performance gap. This increases the number of cycle the processor has to wait for the memory accesses dramatically and results in serious overall performance degradation. It is well known that the smaller caches run faster. Therefore the L1 cache is usually made small in order to keep up with the processor speed. However, low level cache must remain large enough to maintain low global miss rate therefore results in relative slower access speed. Traditional cache architectures assume that each level in the cache hierarchy has a single, uniform access time. Since large low level caches are usually made by combining many smaller SRAM sub-arrays, its access time is determined by the longest access time among all sub-arrays. However, such uniform access fails to exploit the difference in latencies among sub-arrays.

To increase average access speed on large low level cache, a recent work proposed adaptive, non-uniform cache architecture (NUCA) [11]. NUCA exploits the variation in access time by partitioning the large low level cache into several, smaller sub-arrays. Each sub-array has different access time and can be probed independently. Since large low level caches are usually made up of many SRAM sub-arrays which are spread out throughout the chip and are connected through long wires, this partition can be done easily. The basic architecture of NUCA is shown in Figure2.2. NUCA allows fast access to close sub-arrays while maintaining slow access to far sub-arrays. For each cache set, each way is placed among different sub-arrays of different access latency. On cache access, NUCA searches the ways sequentially, accessing both tag and data array, from the fastest way to the slowest way. The search completes when a hit is detected, and subsequent accesses on slower sub-arrays are saved.

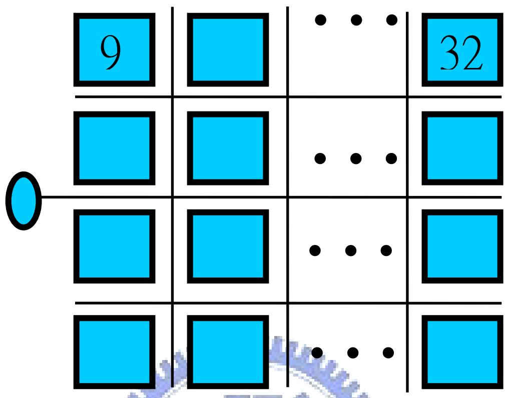


Figure 2.2: NUCA

To increase cache efficiency, NUCA pioneer the concept of data placement by allowing frequently accessed data to be placed in faster ways while infrequently accessed data are placed in slower ways. This is done by allowing the hit block to be swapped to the faster ways within a set. A newly introduced block is initially placed in the slowest way. If a block is accessed frequently, this block will “bubbles” toward the fastest way. On the other hand, an infrequently accessed data will remain in the slower way.

To ensure equal performance for every set in the cache, every sub-array must contain equal number of ways for each set in the cache. This implies that the NUCA can only place a small number of ways for each set in the fastest sub-array. To increase efficiency on the fastest sub-array, Chishti, et al proposed NuRAPID

(Non-uniform access with Replacement And Placement usIng Distance associative) [12] cache which employs the concept of *distance-associative* to completely decouple tag placement with data placement. This is done by adding a *forward pointer* for each tag to indicate the location of its corresponding data block. Distance-associative allows the tag placement to maintain set-associative for accessibility but allows data blocks to be placed in any sub-array without restriction. Therefore, unlike the original NUCA, all the frequently accessed blocks can migrate to the fastest sub-array whether they are in the same set or not. This can increase the hit rate on the fastest sub-array and therefore can reduce average access latency. Distance-associative also enables serial tag-data access, which is commonly used in low level caches to reduce cache power since data array is much larger, therefore slower than tag array in low level cache.

Although intended for reducing access latency on low-level caches, by increasing the hit rate on earlier accessed sub-arrays using smarter placement policies, substantial energy can also be saved by NUCA since subsequent accesses on slower sub-arrays may be saved. NuRAPID further increases the performance by allowing efficient data placement using distance-associative. However, distance-associative requires a forward pointer for each tag entry and therefore large additional memory storage is required for tag array which increases tag array access time.

Chapter 3 Proposed Low-Power Cache Architecture

3.1 Drawbacks of Conventional Set-Associative Cache

The basic architecture of conventional 2-way set-associative cache is shown in figure 3.1. Each way consists of one tag array and one data array. For each tag array, a comparator is required to compare the accessed tag from the tag array with the request tag. A read access first probes all tag arrays and data arrays. The accessed tags are then compared with the request tag to determine if there is a match. If a match is detected, the corresponding data block of the matching way is then selected by the multiplexer to the output. The write access follows similar access procedure as the read access except that data arrays can not be written until a match is detected.

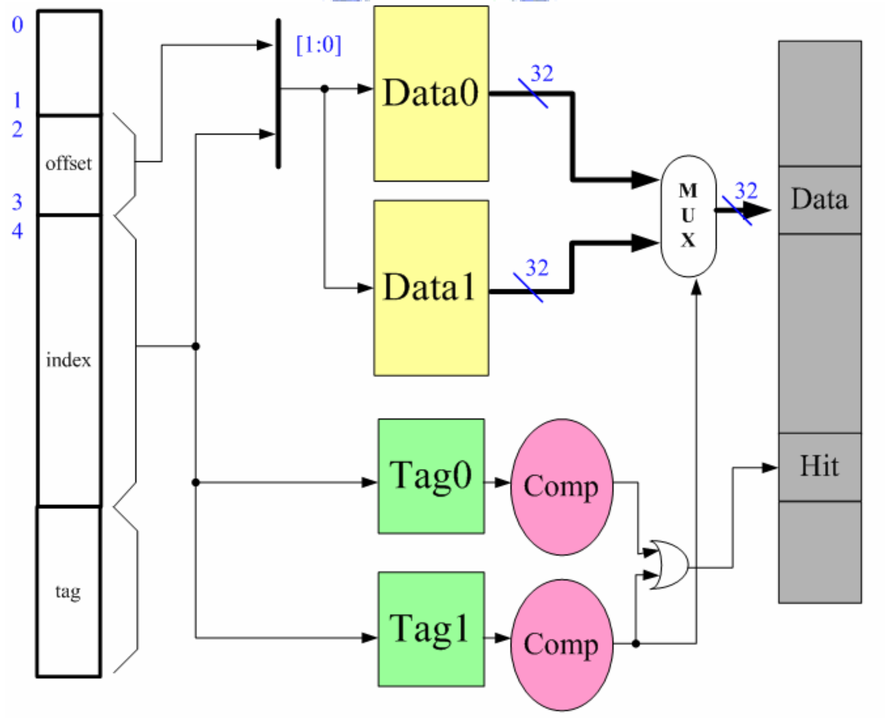


Figure 3.1: Conventional 2-Way Set-Associative Cache

Two problems on set-associative cache are observed. These problems are stated below.

(1) Redundant Memory Access

During a cache access, both ways are accessed, but actually only the matching way is needed. The energy spent accessing the other ways is therefore wasted. Figure 3.2 shows the example on read access which hits in Way 0. It can be seen that the accesses on the tag array and data array of the un-matched way, which is Way 1, have no effect on the result and therefore the energy spent accessing these arrays are wasted. The write access has similar result except that since data array can not be written until a match is detected, only the access on the tag array of the un-matched way is wasted.

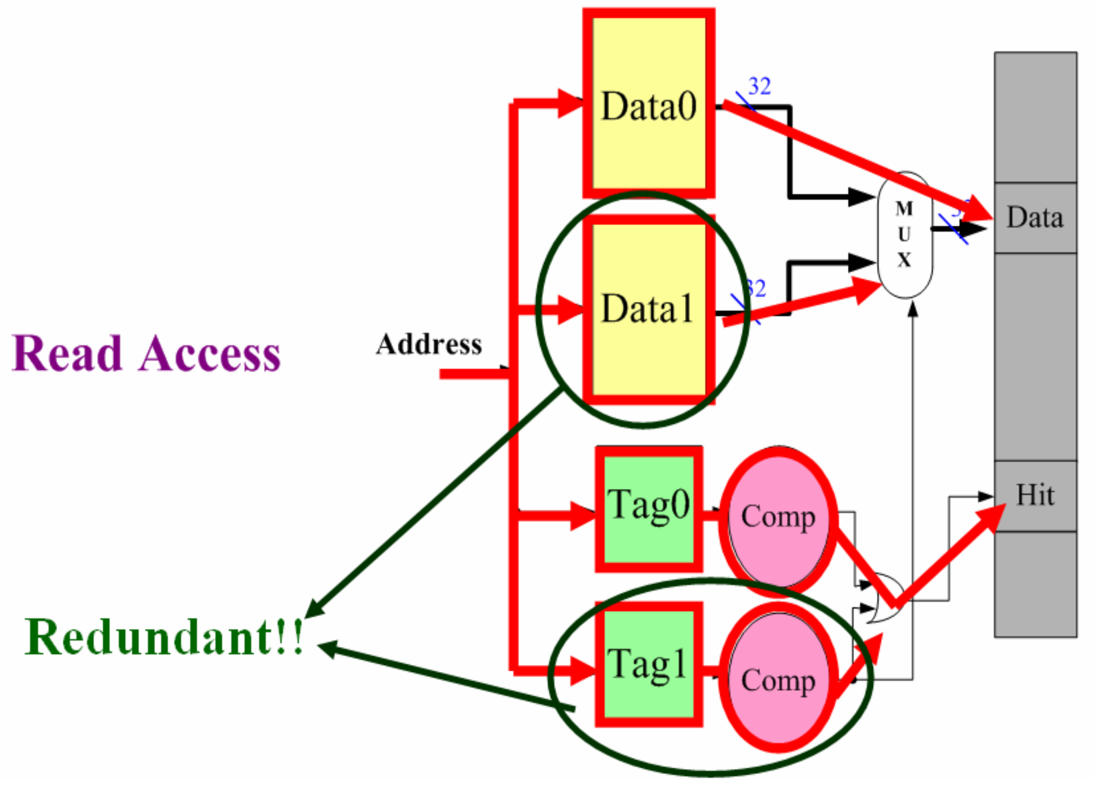


Figure 3.2: Read access on conventional set-associative cache

(2) Hit-Signal Fanout

The second problem concerns about the fanout on the hit-signal. The hit-signal generally refers to the outputs of the tag comparators in the cache to indicate if a way hits. In set-associative cache, a multiplexer is required to select the hit data among all the ways to the output. Since the size of data blocks are usually large (32-bit or 64-bit for recent processors), the loading of this multiplexer is usually quit large. To achieve one cycle hit time, conventional set-associative cache accesses tag arrays and data arrays in parallel. The hit-signal determined by comparator is then used as the select signal of the multiplexer that used to select hit data block among all the ways. Therefore the multiplexer causes a large fanout on the hit-signal as shown in figure 3.3. Although tag arrays are usually smaller than data arrays and therefore have faster access time, the total access time on tag array plus its subsequent comparator, which determines the hit-signal, is usually longer than the access time on data array and therefore contributes to the critical path of set-associative cache.

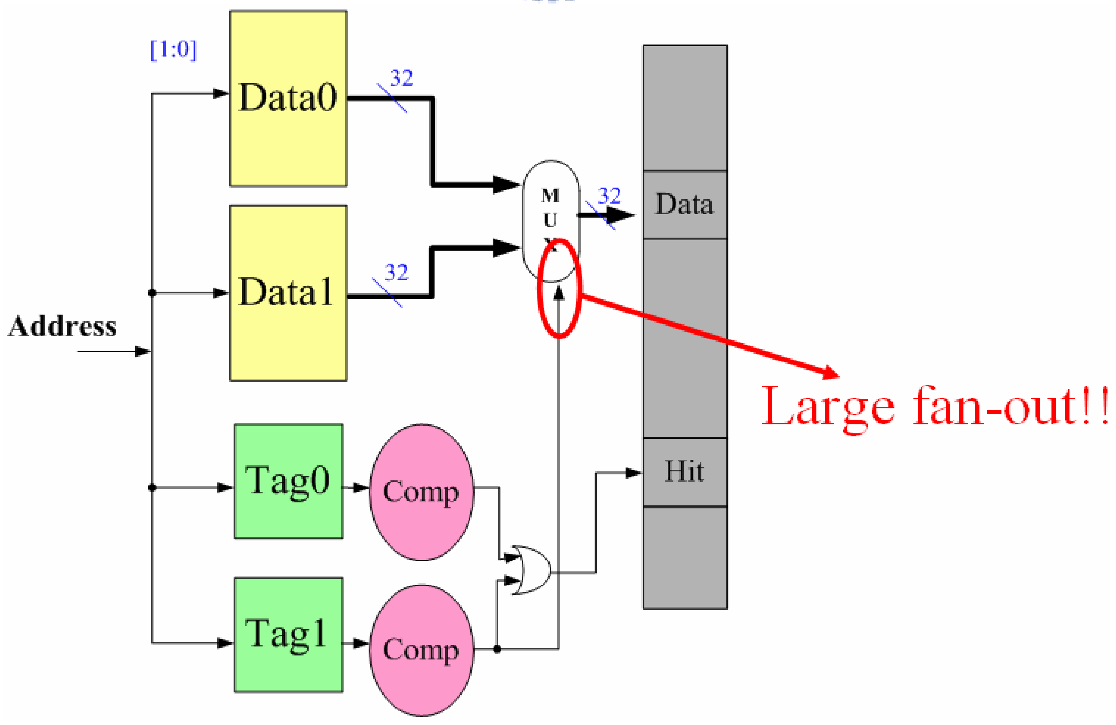


Figure 3.3: Large fanout on the hit-signal

To understand the effect of the fanout of the hit-signal on the critical path of set-associative cache, the access times on several sub-structures of a 2-way set-associative cache are derived. The sub-structures that are examined are shown in figure 3.4, figure 3.5, and figure 3.6. In Figure 3.4, the access time of the two basic structures of a conventional direct-mapped cache, which are the data array (**data**) and the tag array plus comparator (**tag+comp**), are examined. In Figure 3.5, the access time of the way-matching part (**tag+comp+o**r) and the data accessing part (**data+mux**) of the conventional 2-way set-associative cache are examined. Finally, the basic structure of conventional 2-way set-associative cache, which is formed by connecting the hit-signal derived from (**tag+comp+or**) to the select signal of the multiplexer in (**data+mux**) of figure 3.5, is examined as shown in figure 3.6.

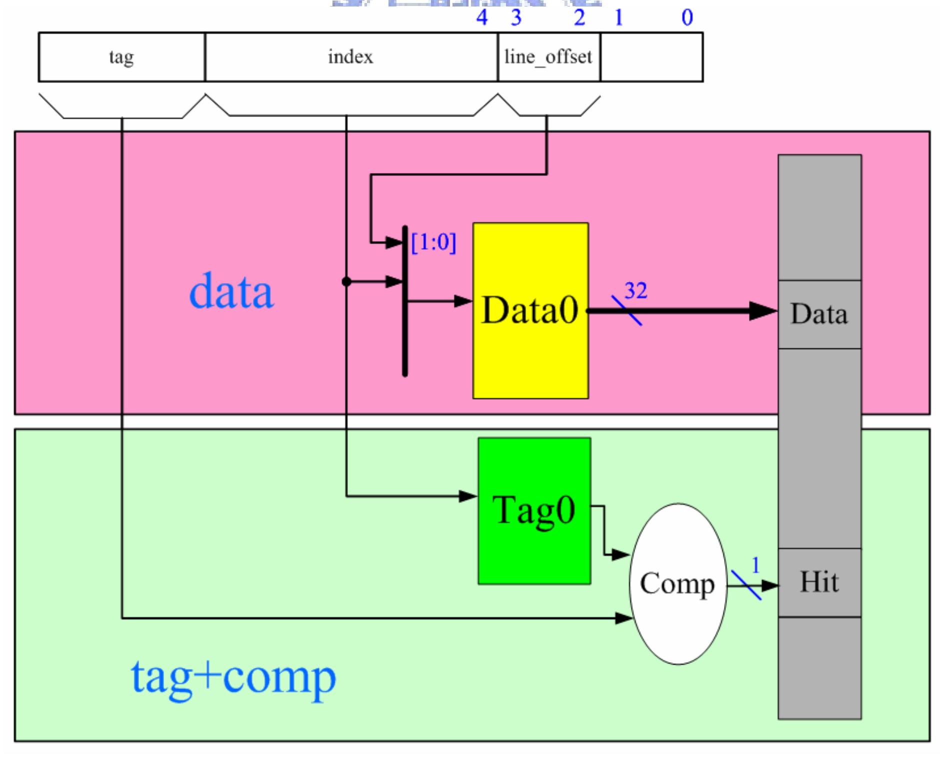


Figure 3.4: Decomposition of conventional direct-mapped cache

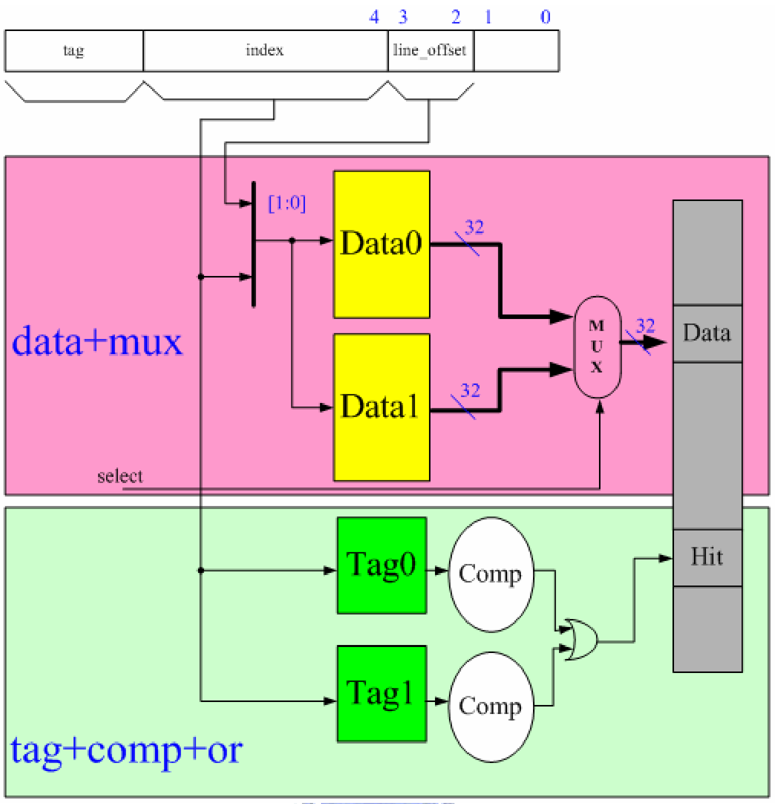


Figure 3.5: Decomposition of conventional set-associative cache

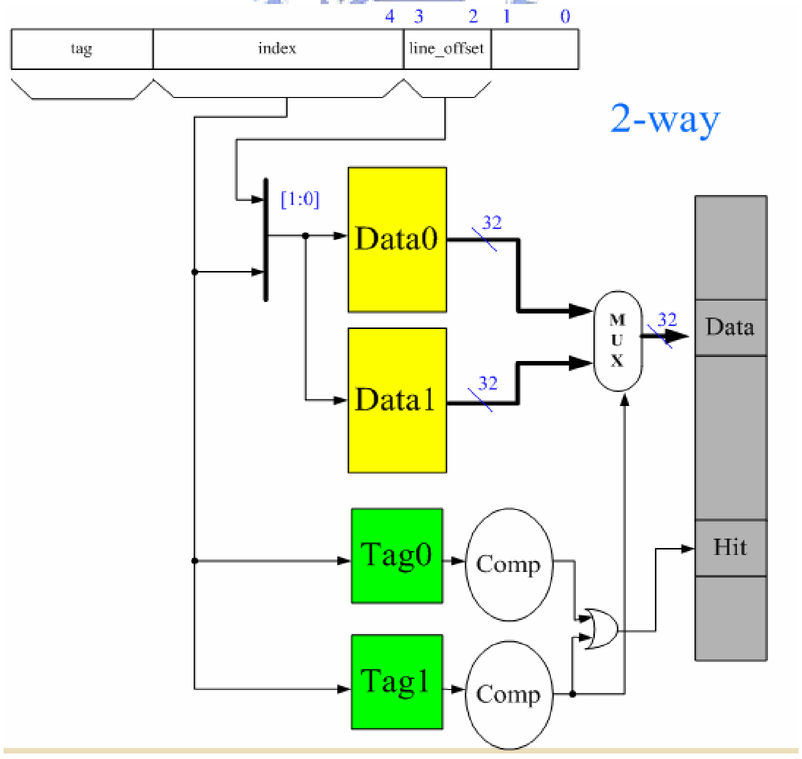


Figure 3.6: Basic architecture of conventional 2-way set-associative cache

Table 3.1 shows the access time of sub-structures of 2-way set-associative cache. The memory cells are generated using Artisan UMC 0.18um memory compiler and the access times are determined using Synopsys 0.18um design compiler. The size of each data block is 32 bytes and each cache line contains 4 blocks. The minimum and maximum sizes of memory cell the memory compiler can generate are 1KB and 32KB. (**data+mux**), (**tag+comp+or**), and (**2_way**) require 2 individual memory cells for each way therefore the minimum size available for these three structures are 2KB.

cache size	data	tag+comp	data+mux	tag+comp+or	2_way
1k	2.29	3.19	—	—	—
2k	2.3	3.17	2.51	3.24	3.74
4k	2.33	3.16	2.52	3.23	3.74
8k	2.38	3.15	2.55	3.22	3.72
16k	2.41	3.14	2.6	3.21	3.7
32k	2.97	3.17	2.63	3.21	3.73
64k	—	—	3.18	3.24	3.73

Table 3.1 Access time of sub-structures of 2-way set-associative cache

By comparing the access time of (**data**) and (**tag+comp**), it can be seen that the tag access part has longer access time than the data access part and therefore contributes to the critical path of direct-mapped cache. The situation remains the same on set-associative cache by comparing the access time of (**data+mux**) and (**tag+comp+or**), even though a multiplexer is added to the data access part. These results reveal that the critical path of conventional set-associative cache is the tag access part plus the multiplexer that selects the hit data block using the hit-signal derived from the tag access part as select signal.

The delay of the multiplexer in (**data+mux**) can be derived by subtracting the access time of (**data+mux**) by the access time of (**data**), which is about 0.2ns. While the delay of the same multiplexer in (**2-way**), which is the access time difference between (**2-way**) and (**tag+comp**) increase to about 0.55ns. The large fanout on the hit-signal due to the multiplexer weaken its driving ability, results in increases the access time on the multiplexer and therefore increases the overall cache access cycle time

3.2 Sequential Way Access

Phase cache reduces the number of access by accessing tag arrays first to determine the matching way and then accesses only the data array of the matched way if there is a hit. However, since serial tag-data access has the wait until the matching way is known before the access on data array can begin, cache access latency on each cache access is therefore increased.

To reduce redundant memory accesses while maintaining fast cache access clock rate, we propose sequential way-access set-associative cache which searches each way sequentially, accessing both the tag array and the data array of the searched way, from the first way to the last way. The access sequence terminates when a match is detected and subsequent accesses on the remained ways are then saved. Unlike the phase cache, which reduces the number of redundant memory accesses by reducing the number of data arrays accessed, sequential way set-associative cache reduces the number of redundant memory accesses by reducing the number of redundant way accesses. Since both tag and data array are probed in parallel for each way access, the access latency for each way is about the same as conventional set-associative cache. By letting each way access to take one cycle, the cache access clock rate of sequential way-access set-associative cache can be compatible with the conventional set-associative cache.

Figure 3.7 shows an example of 2-way sequential way-access set-associative cache. During the first cycle of cache access, only the tag array and the data array of way 0 are probed. If way 0 hits, as shown in Figure 3.7(a), the cache access is completed in one cycle as the same as conventional 2-way set-associative cache and the accesses on way 1 are saved. If way 0 misses, subsequent accesses on way 1 are required in the next cycle as shown in Figure 3.7(b). In this case, no memory accesses are saved compared to conventional 2-way set-associative cache. Further more, one additional access cycle is required no matter way 1 hits or misses. Overall, sequential way set-associative cache reduces the number of memory access at the expense of the increase in average memory access time.

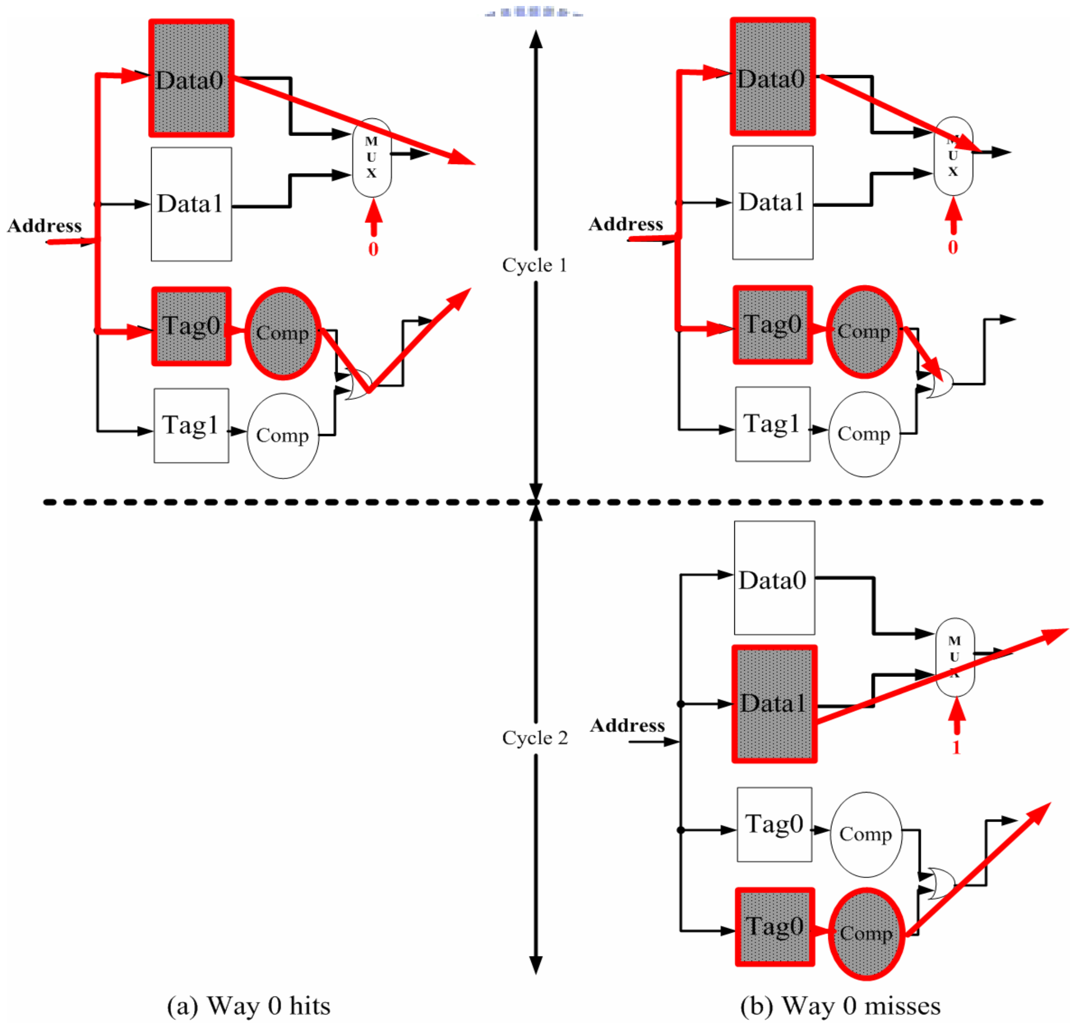


Figure 3.7: 2-way sequential way-access set-associative cache

The average memory access time can be expressed as shown in formula (3). Due to the ever widen gap between the process clock rate and main memory access time, the miss penalty continues to grow as a dominant factor in average memory access time. Therefore, although sequential way-access set-associative cache increases the average hit time, this overhead on AMAT should be minor compared to the large miss penalty when cache misses.

$$\text{AMAT} = (\#\_\text{of}\_\text{access}) \times (\text{hit}\_\text{rate} \times \text{hit}\_\text{time}_{\text{cycle}} + \text{miss}\_\text{rate} \times \text{miss}\_\text{penalty}) \times \text{clk} \quad (3)$$

Besides reducing the number of memory access, sequential way-access set-associative cache also provides a mean to reduce the fanout on the hit-signal. Since the accessed way is predetermined on each memory cycle, the select signal of the multiplexer that used to select hit data block among all the ways can also be determined in advance as shown in Figure 3.7. Therefore the fanout on the hit-signal to the data multiplexer in conventional set-associative cache is no longer required in sequential way-access set-associative cache and large fanout on the hit-signal can be eliminated. Since the hit-signal is the most dominant factor on the actions of a cache, the largely reduced fanout on the hit-signal may allows more complex cache access schemes and placement policies without increasing the overall fanout on the hit-signal, and therefore the cache access cycle time, dramatically. Further more, the cache access cycle time may even reduce if the increased fanout on the hit-signal is less than the reduced fanout caused by the data multiplexer.

3.3 Placement and Replacement Policy

Summarizing the overheads on cache hit on each way in sequential way-access set-associative cache, we can see that a cache hit on different ways of sequential way-access set-associative cache has different amount of overhead as opposed to conventional set-associative cache. A cache hit in the earlier accessed ways can eliminate subsequent accesses on later ways. Further more, less additional overheads in cycle counts are required. The optimum situation comes when the cache hits in the earliest accessed way. In this case only the hit way is activated and no additional access cycle requires as the same as conventional set-associative cache. On the other hand, the later a cache hits, the less subsequent way accesses can be saved while more additional access cycles are required. The worst situation comes when the cache hits in the last accessed way or cache misses. In these cases no memory access is saved but causes n-1 additional access cycles where n equals to the associativity of the cache. Therefore, the earlier accessed ways have less access overhead than later accessed ways.

According to the above conclusion, a possible way to increase the performance of sequential way-access set-associative cache is to increase the hit rate on the earliest accessed way. By using smarter placement and replacement policies, we can allow more frequently accessed blocks to be placed in earliest accessed ways while infrequently accessed data are placed in slower accessed ways. Several approaches have been proposed to increase the cache hit rate on frequently accessed blocks [15, 16, 17, 18]. Although complex placement and replacement policies cause additional fanout on the hit-signal, the eliminated fanout on the hit-signal caused by data multiplexer in conventional set-associative cache can compromise this overhead.

The basic principle that is applied on the placement and replacement policies of sequential way-access set-associative cache is temporal locality. Temporal locality basically means that if a memory location is recently referenced, there is a high probability that the same location will be accessed again soon. Therefore the goal is to reduce the access overhead on recently accessed blocks in order to increase the performance. This can be done by allowing recently accessed blocks to be placed in the earliest accessed way.

(1) Priority Replacement Policy

The priority replacement policy used in sequential way-access set-associative cache concerns of the decision of which cache line to be evicted and the placement of the introduced cache line when a cache miss occurs. During a cache miss, conventional set-associative cache applies conventional least-recently-used (LRU) replacement policy to determine which block within the selected set is least recently accessed and therefore the cache line containing that block should be evicted and replaced by the newly introduced cache line containing the refilled block. In priority replacement policy used in sequential way-access set-associative cache, the LRU detection unit is preserved to select the evicted block within the selected set on cache miss. However, in order to reduce the re-access overhead on the refilled block, priority replacement policy requires to place the refilled blocks in the earliest accessed way within the selected set. Figure 3.8 shows an example of priority replacement policy on a 2-way sequential way-access set-associative cache, assuming the LRU block of the selected set is in the earliest accessed way (Way 0). Since the way of the evicted block (A) is exactly the same as the way the newly introduced block (C) should be placed, the space created by dropping the cache line containing the evicted block in the earliest accessed way is refilled by the cache line containing the refilled block. The result is exactly the same as conventional LRU policy used in conventional set-associative cache and no additional overheads are required.

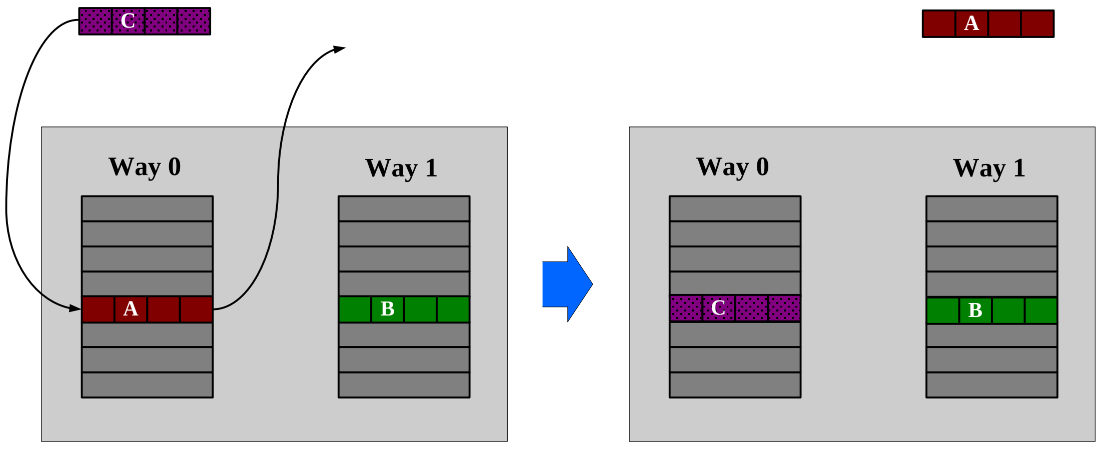


Figure 3.8: Priority replacement policy (LRU : Way 0)

On the other hand, if the LRU block of the selected set is in the later accessed way, additional memory accesses are required to create a space in the earliest accessed way for the newly introduced cache line containing the refilled block. Figure 3.9 shows another example of priority replacement policy on 2-way sequential way-access set-associative cache. This time we assume the LRU block of the selected set is in the later accessed way (Way 1). In this case the eviction of the cache line containing the LRU block (B) creates a space in Way 1 while the newly introduced cache line containing the refilled block (C) suffers from no available space in the earliest way for the cache line to be placed. To solve this problem, priority replacement policy requires that the cache line containing the block that is originally in the earliest accessed way of the selected set (A) to be moved to the way where the eviction occurs in order to create a space for the newly introduced cache line in the earliest accessed way. Although moving cache line between ways causes additional memory accesses compared to conventional LRU replacement policy, especially for d-cache since a cache line usually contains more than one block. The reduced memory accesses on re-accessing the refilled blocks can compromise the increased accessed. Further more, the access cycles on re-accessing the refilled block can also be reduced.

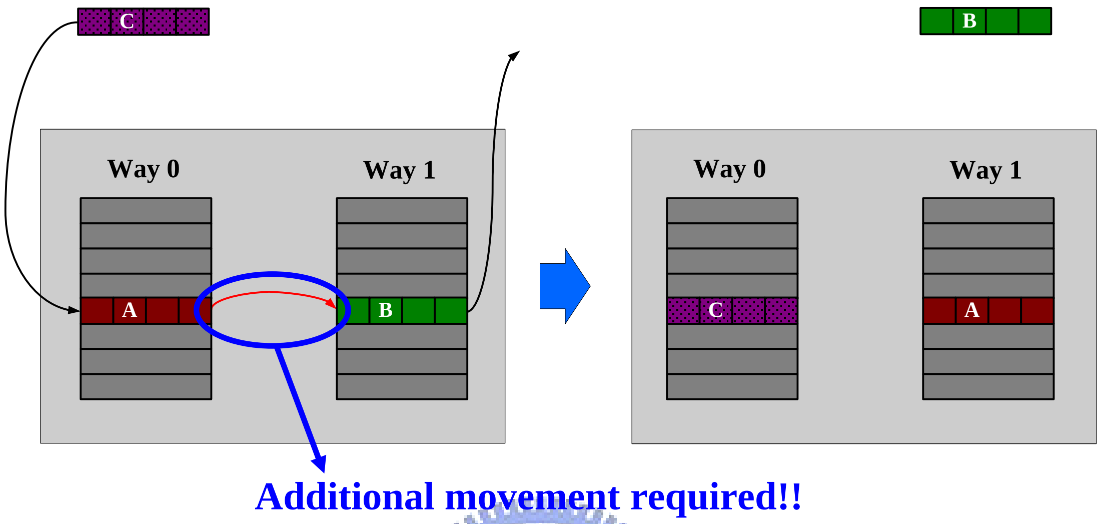


Figure 3.9: Priority replacement policy (LRU : Way 1)

(2) Promotion Placement Policy

To further increase the performance of sequential way-access set-associative cache, a possible solution is to reduce the access overhead on recently accessed block during cache hits. In promotion placement policy used in sequential way-access set-associative cache, during a cache hit, the hit block is required to be promoted to the earliest accessed way in order to reduce the re-access overhead on the hit block. When the cache hits in the earliest accessed way, no additional action is required since the hit block is already in the earliest accessed way. On the other hand, when the cache hits in a way other than the earliest accessed way, swap is required to promote the cache line containing the hit block to the earliest accessed way while the cache line containing the block that is originally in the earliest accessed way of the selected set is placed to the hit way. This is shown in figure 3.10, assuming the cache hits in the later accessed Way 1 of sequential way-access set-associative cache.

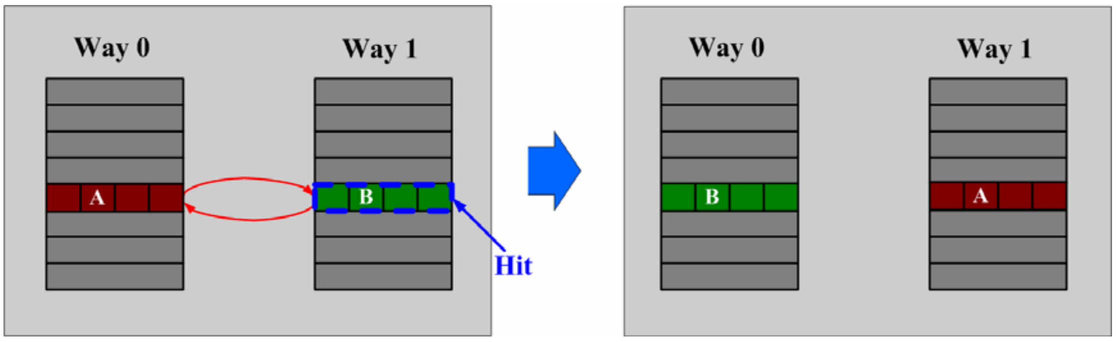


Figure 3.10: Promotion placement policy (hits in Way 1)

Besides reducing the re-access overhead on recently accessed block, promotion placement policy also provides a mean to prevent frequently accessed blocks from becoming stuck in the later accessed ways. Consider the following access sequence shown in figure 3.11, assuming block-A, block-B, and block-C all map to the same set. According to the access sequence we can see that block-A is a frequently accessed block while block-B is not. The first access on block-A puts block-A to the earliest accessed way according to the priority replacement policy and therefore allowing the following accesses on block-A to have minimum access overhead. If however, a glitch access on the infrequently accessed block-B occurs. According to the priority replacement policy, block-B will be placed to the earliest accessed way while frequently accessed block-A will be demoted to the later accessed way. Without promotion placement policy, frequently accessed block-A will be stuck in later accessed way therefore will increase the access overhead on following accesses on block-A. Promotion placement policy allows accidentally demoted block-A to be swapped back to the earliest accessed way and therefore can prevent the access overhead on subsequent accesses on block-A from increasing.

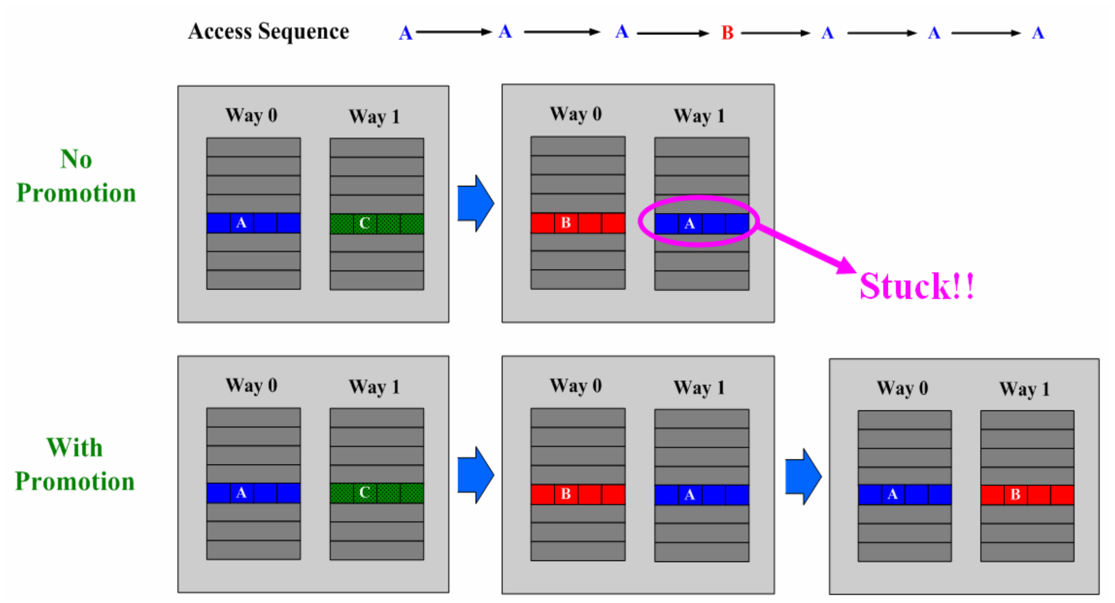


Figure 3.11: Promotion on frequently accessed block

Chapter 4 Experimental Evaluation

4.1 Experimental Environment

Figure 4.1 shows the basic simulation framework used for experimental evaluation. The memory activity trace files are derived by running benchmarks on an ARM7 compatible processor developed by our laboratory called ACARM7. The parameters of ACARM7 are listed in Table 4.1. The memory activity information of the benchmarks running ACARM7 is extracted and recorded in the memory activity trace files. Both the activities of I-cache and D-cache are extracted by separating the requests for instruction and data. The memory activity trace files are then served as the processor core for the cache during the simulation.

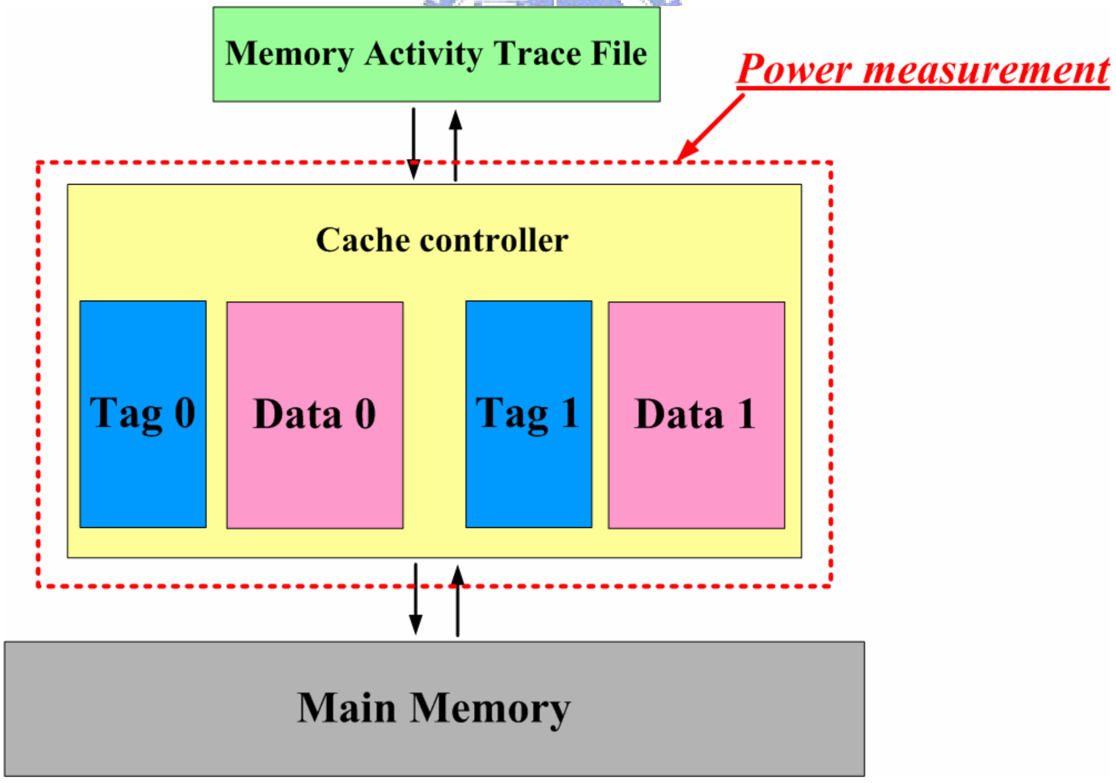


Figure 4.1: Simulation framework

ISA	ARM v4 ISA
Issue Width	32 bits
Pipeline	3 stages
Architecture	von Neumann architecture

Table 4.1: Parameters of ACARM7

The base line cache architecture is a 2-way set-associative cache of size from 2KB to 128KB. The issue size for each cache access is 4 bytes. The cache block size is 16 bytes. For comparison purposes, four cache configurations are developed: they are conventional set-associative cache (***2 Way Conv***), sequential way-access set-associative cache without any optimization in placement (***Seq 2 Way***), sequential way-access set-associative cache with priority replacement policy (***Seq+Pri***), and sequential way-access set-associative cache with both priority replacement policy and promotion placement policy (***Seq+Pri+Pmt***). All cache models are developed using hardware description language (Verilog) and are synthesized using Artisan UMC 0.18um technology. The memory cells are generated using Artisan UMC 0.18um memory compiler. The minimum and maximum sizes of memory cell the memory compiler can generate are 1KB and 32KB. Concerning the access time of a 32KB memory cell increases too much compared to memory cells of other size (see first column on table 3.1), each way on the 128KB cache is made up of four 16KB memory cells instead of two 32KB memory cells and only the selected cell on each way is activated on each cache access. The power measurement includes both the cache controller and the memory arrays used by the cache, including tag arrays and data arrays. The power is measured using power compiler with clock rate fixed at 200MHz on worst case.

The main memory is basically a behavior model which is used as a backup storage for the cache. The issue size of the main memory is 16 bytes. A write transfer takes 18 cycles and a read transfer takes 16 cycles to complete.

The EDA environment used in the simulation are summarized in table 4.1.

Technology	UMC 0.18um
Simulator	NC Verilog
Synthesis	Design compiler
Power analysis	Power compiler
Memory block	Artisan UMC 0.18um memory compiler

Table 4.2: EDA environment

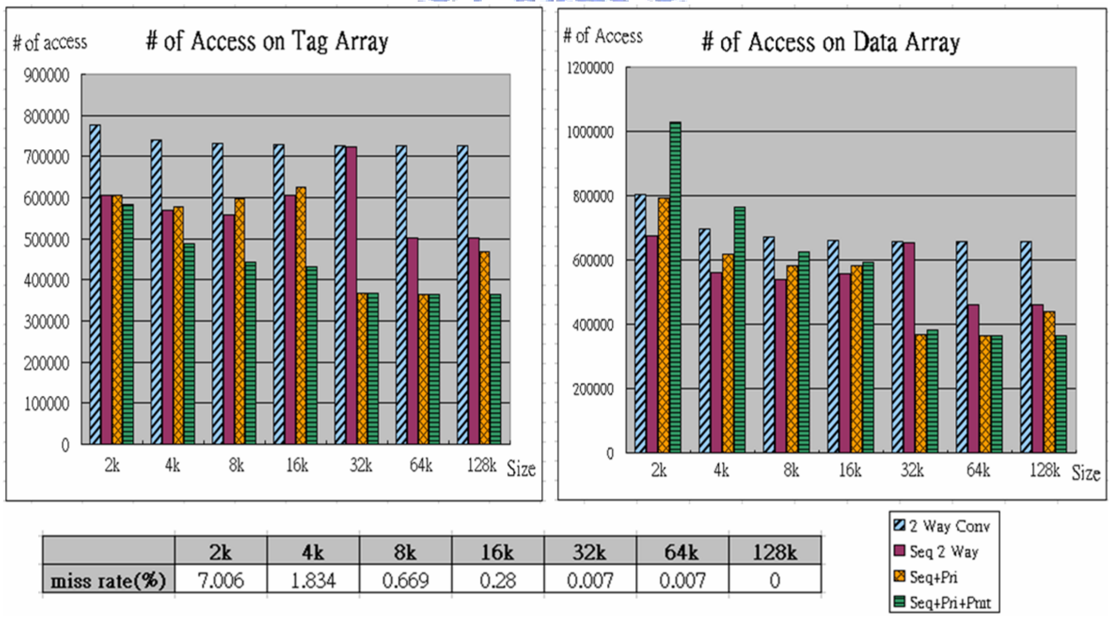
Six benchmarks are simulated for each cache configuration. The first one is a quick sort program of 65536 elements. The second and the third are JPEG encoder and decoder respectively. The encoder converts a 64*64 bitmap file to 64*64 jpeg file and the decoder converts the 64*64 jpeg file back to 64*64 bitmap file. The fourth one is the Whetstone benchmark. The last two are FFT and Matrix from the Dspstone benchmark suite. The FFT contains 1024 data and the dimension of the matrix is 50X50.

4.2 Experimental Result

4.2.1 Memory activity

Figure 4.2 shows the memory activities of JPEG encoder on D-caches of different cache configurations. Both the activities on tag array and data array are presented. The corresponding miss rates for each cache size are also presented. Since for each cache size, both the capacity and associativity on each cache configuration are the same, the miss rates for each cache configuration on the same cache size are the same.

From the figure, it is clearly that sequential way access (***Seq 2 Way***) does reduce the number of memory access. However, the reduction rate is not quit stable against miss rate. The number of access actually increases as cache size increases from 4K to 32K while the miss rates are reduced. The reason is because the placement on sequential way access alone depends only on the conventional LRU replacement policy. A frequently accessed block may be placed in the later accessed way if the replaced block is in the later accessed way. Since there is no mechanism to move blocks between ways, the frequently accessed block may become stuck in the later accessed way and compromise the benefit of sequential way access.



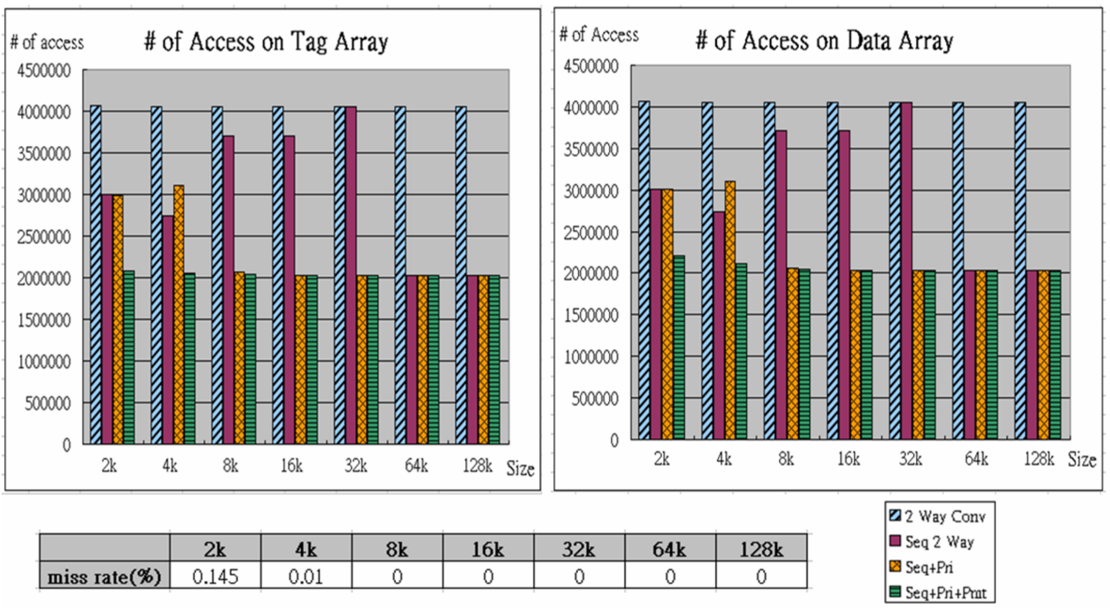
	2k	4k	8k	16k	32k	64k	128k
miss rate(%)	7.006	1.834	0.669	0.28	0.007	0.007	0

Figure 4.2: Memory activities on D-cache running JPEG encoder

Adding priority replacement policy to sequential way access allows newly introduced blocks to be placed in the earliest accessed way therefore reduces the re-access overhead on newly introduced blocks. However, priority replacement policy only applies during cache miss. A frequently accessed block which is originally in the earliest accessed way may be accidentally demoted to the later accessed way when a new block is introduced to the cache and become stuck there. Therefore the access reduction rate on ***Seq+Pri*** against miss rate is smoother comparing to ***Seq 2 Way*** but still not quit stable. Further more, moving blocks between ways cause additional memory accesses and might compromise the reduced accesses benefited from priority replacement policy.

By allowing frequently accessed blocks in the cache to be placed to the earliest accessed way during cache hit, ***Seq+Pri+Pmt*** effectively reduces the number of memory access compared to conventional set-associative cache as miss rate reduces. For tag array, sequential way access with both priority replacement policy and promotion placement policy has the lowest number of access against all other cache configurations for all cache sizes. For the data array, since a cache line contains four blocks, a cache line swap requires 8 accesses to read out the two cache lines to be swapped and 8 accesses to write the cache lines to the required destination, resulting 16 accesses for each swap on data array comparing to only 4 accesses on tag array. Therefore when the cache size is small, the high probability of swaps due to thrashing on ***Seq+Pri*** and ***Seq+Pri+Pmt*** increases their number of access on data array more dramatically than on tag array. However, as cache size increases, the ease on capacity miss allows the benefits of swaps on ***Seq+Pri*** and ***Seq+Pri+Pmt*** to show, resulting more serious reduction in the number of access for both tag array and data array, especially for ***Seq+Pri+Pmt***.

Figure 4.3 shows the memory activities of the same JPEG encoder on I-caches of different cache configurations. The results are similar to the D-cache. The figures also shows that up to 50% of accesses can be saved on 2-way sequential way-access set-associative cache compared to conventional 2-way set-associative cache. The effects on priority replacement policy and promotion placement policy are more evidence in the figure. Without the help of smarter placement/replacement policies, ***Seq 2 Way*** achieves about 50% of access reduction when the cache size increase to 64KB, even though the miss rate is already 0% when cache size reaches 8KB. ***Seq+Pri*** achieves about 50% reduction much earlier than ***Seq 2 Way*** at the cache size of 8KB, the same size when the miss rate achieves 0%. For ***Seq+Pri+Pmt***, almost 50% of access reduction is achieved at the cache size of 2KB, despite the fact that the miss rate is not yet 0%.



	2k	4k	8k	16k	32k	64k	128k
miss rate(%)	0.145	0.01	0	0	0	0	0

Figure 4.3: Memory activities on D-cache running JPEG encoder

The memory activities of each benchmark on 32-KB D-caches of different cache configurations are shown in table 4.3 and table 4.4 where table 4.3 shows the activities on tag arrays and table 4.4 shows the activities on data arrays. Sequential way access in average reduces the number of access on both tag arrays and data arrays effectively. Priority replacement policy allows newly introduced block to be placed in the earliest accessed way during cache miss and further reduces the number of access.

Combining with promotion placement policy, which places recently accessed blocks in the earliest accessed during cache hit, ***Seq+Pri+Pmt*** achieves the highest reduction among all cache configurations. Up to 43% of accesses are reduced on tag arrays and 22% on data arrays. For the data arrays, the large cache line increases the overhead on swaps consequently compromises the benefits of priority replacement policy and promotion placement policy.

32KB D-cache (16-byte block)							
	# of access on tag array						Average Reduction(%)
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	
2 Way Conv	10877110	726582	2544710	542078	382328	1050558	—
Seq 2 Way	7339463	721974	2077295	278209	295676	853405	23.61
Seq+Pri	8620797	367145	2065885	278285	203443	852207	33.89
Seq+Pri+Pmt	6041373	368539	2023675	271321	201806	559923	43.01

Table 4.3: Memory activities of benchmarks on tag arrays of 32-KB D-caches

32KB D-cache (16-byte block)							
	# of access on data array						Average Reduction(%)
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	
2 Way Conv	10112040	656508	2973387	429463	303624	900961	—
Seq 2 Way	7604913	652035	2611631	278285	249111	730744	18.28
Seq+Pri	8703385	367358	3797953	278589	203422	730580	19.55
Seq+Pri+Pmt	7505161	381091	3644583	271937	226558	642979	22.64

Table 4.4: Memory activities of benchmarks on data arrays of 32-KB D-caches

4.2.2 Power

Table 4.5 shows the power consumptions of JPEG encoder on D-caches of different cache configurations. The power reduction rate of each cache configuration correspond to the conventional set-associative cache of the same cache size are shown in figure 4.4. The curve of power reduction rate against the cache size is similar to the fraction of reduced access. It is clearly that the reduced number of access due to sequential way access effectively reduces cache power consumption.

Power (mW)	2k	4k	8k	16k	32k	64k	128k
2_way_Conv	10.8849	11.1911	11.3681	11.753	12.7075	14.1401	13.4773
Seq 2 Way	8.6416	8.5862	8.7266	9.2643	11.0202	9.9955	9.5769
Seq+Pri	10.2043	9.656	9.6768	10.0554	8.4002	9.1785	8.5338
Seq+Pri+Pmt	11.3855	10.2995	9.4004	9.4849	7.9818	8.5338	8.1694

Table 4.5: Power consumptions on D-cache running JPEG encoder

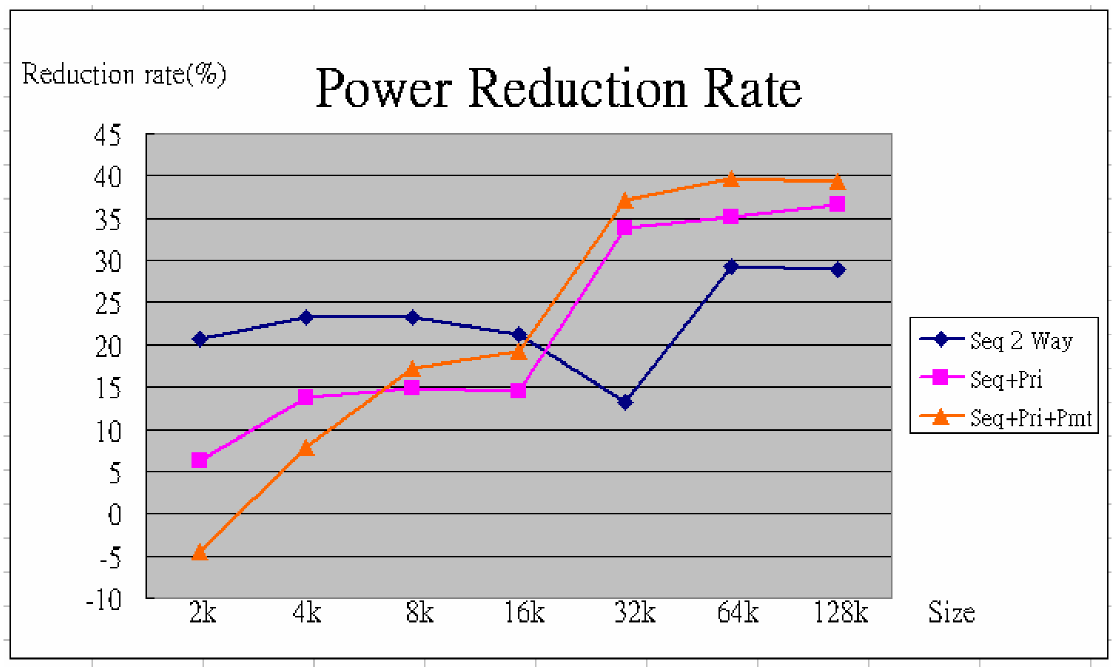


Figure 4.4: Power reduction rate on D-caches running JPEG encoder

Table 4.6 and figure 4.4 also shows the corresponding power consumption and power reduction rate of JPEG encoder on I-caches of different cache configurations. Again, similar results are shown. From the figure it can be seen that under the same number of reduced accesses, ***Seq+Pri+Pmt*** performs better compared to other cache configurations. It is because the LRU detection unit in ***Seq+Pri+Pmt*** is eliminated and therefore further reduces its hardware complexity. The detail will be stated in section

4.2.4. For ***Seq 2 Way***, even though virtually no memory accesses are saved for cache size from 8K to 32K according to figure 4.3, large power reduction is still achieved. The almost perfect hit rate implies that almost every cache access on ***Seq 2 Way*** hits but hits in the later accessed way. Therefore the average cache access time on ***Seq 2 Way*** almost doubles and therefore flattens the energy consumed in each second.

Power (mW)	2k	4k	8k	16k	32k	64k	128k
2_way_Conv	57.9348	58.8702	59.5442	61.572	67.3354	75.9281	70.0092
Seq 2 Way	32.586	32.6055	33.9831	35.0833	38.634	40.3688	37.7288
Seq+Pri	33.2691	33.6019	32.4495	33.4206	36.4221	40.9177	38.3142
Seq+Pri+Pmt	30.1071	29.9822	29.9823	30.8303	33.6149	37.7199	34.597

Table 4.6: Power consumptions on I-cache running JPEG encoder

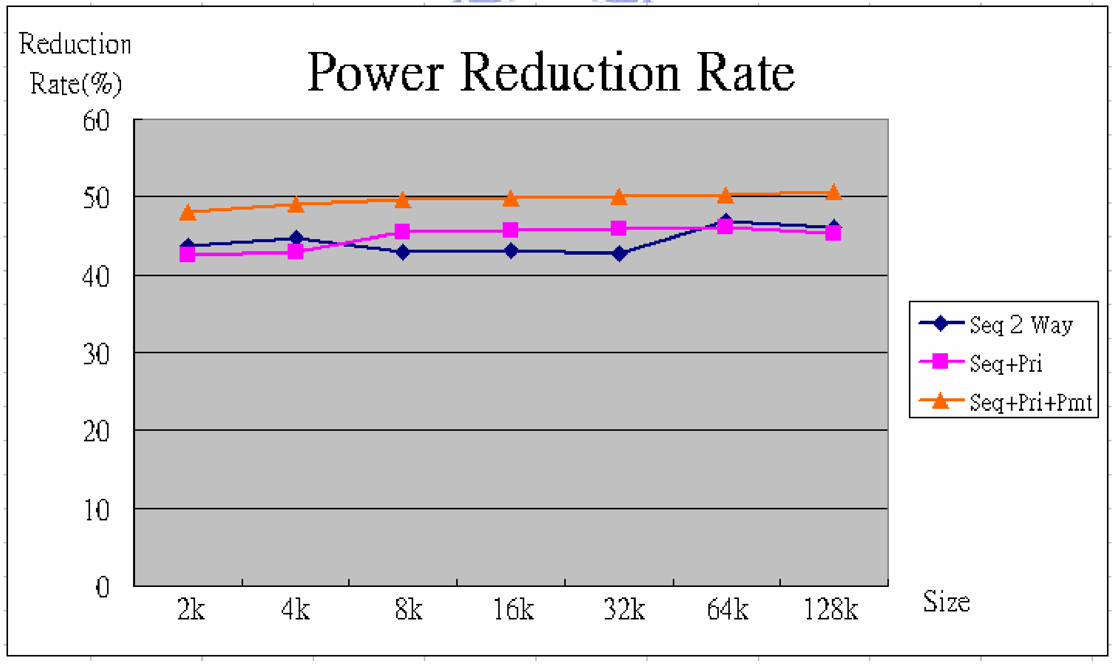
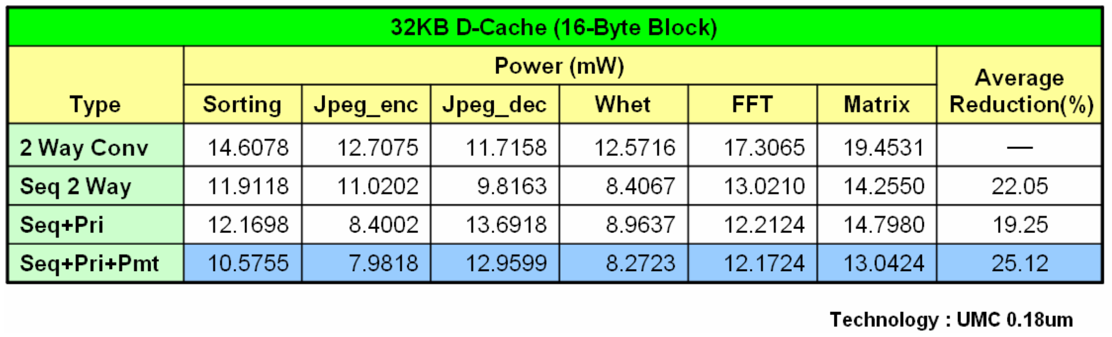


Figure 4.5: Power reduction rate on I-caches running JPEG encoder

Table 4.7 shows the power consumptions of each benchmark on 32-KB D-caches of different cache configurations. For ***Seq+Pri*** and ***Seq+Pri+Pm***t, the hardware overhead due to smarter placement/replacement policies cause additional power consumption and compromise the reduced access power. Despite of that, ***Seq+Pri+Pm***t still manage to achieve the highest power consumption among all cache configurations. Up to 25% of power consumption is reduced by ***Seq+Pri+Pmt*** compared to conventional set-associative cache.



32KB D-Cache (16-Byte Block)							
	Power (mW)						
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	Average Reduction(%)
2 Way Conv	14.6078	12.7075	11.7158	12.5716	17.3065	19.4531	—
Seq 2 Way	11.9118	11.0202	9.8163	8.4067	13.0210	14.2550	22.05
Seq+Pri	12.1698	8.4002	13.6918	8.9637	12.2124	14.7980	19.25
Seq+Pri+Pmt	10.5755	7.9818	12.9599	8.2723	12.1724	13.0424	25.12

Technology : UMC 0.18um

Table 4.7: Power consumptions of benchmarks on 32-KB D-caches

4.2.3 Cycle Count

To evaluate the effects of different main memory latencies on the cycle counts of different cache configurations, two types of main memory with different access latencies are applied. The first one is the same main memory used in pervious evaluations and is referred to as ***mem_long*** in the thesis. The ***mem_long*** requires 18 cycles for a write transfer and 16 cycles for a read transfer. The second main memory has shorter access latency and is referred to as ***mem_short***. A write transfer to ***mem_short*** takes 8 cycles and a read transfer takes 6 cycles. The cycle counts with respect to ***mem_short*** and ***mem_long*** on D-caches of each cache configuration running JPEG encoder are shown in table 4.8 and table 4.9. The percentage of increased cycle count of each cache configuration correspond to conventional set-associative cache of the same size are shown in figure 4.6 (***mem_short***) and figure 4.7 (***mem_long***).

cycle count	2k	4k	8k	16k	32k	64k	128k
2_way_Conv	740474	464650	399602	378312	363642	363642	363264
Seq 2 Way	940689	659542	590068	618980	722298	502734	502329
Seq+Pri	892861	655121	623992	636077	367415	363669	468032
Seq+Pri+Pmt	968675	669817	549477	518382	374977	363669	363561

Table 4.8: Cycle count on D-cache running JPEG encoder (***mem_short***)

cycle count	2k	4k	8k	16k	32k	64k	128k
2_way_Conv	1047274	551590	429672	390482	363912	363912	363264
Seq 2 Way	1266949	754172	623018	630210	722568	503004	502329
Seq+Pri	1199661	742061	654062	647287	367685	363939	468032
Seq+Pri+Pmt	1275475	756757	579547	529592	375247	363939	363561

Table 4.9: Cycle count on D-cache running JPEG encoder (***mem_long***)

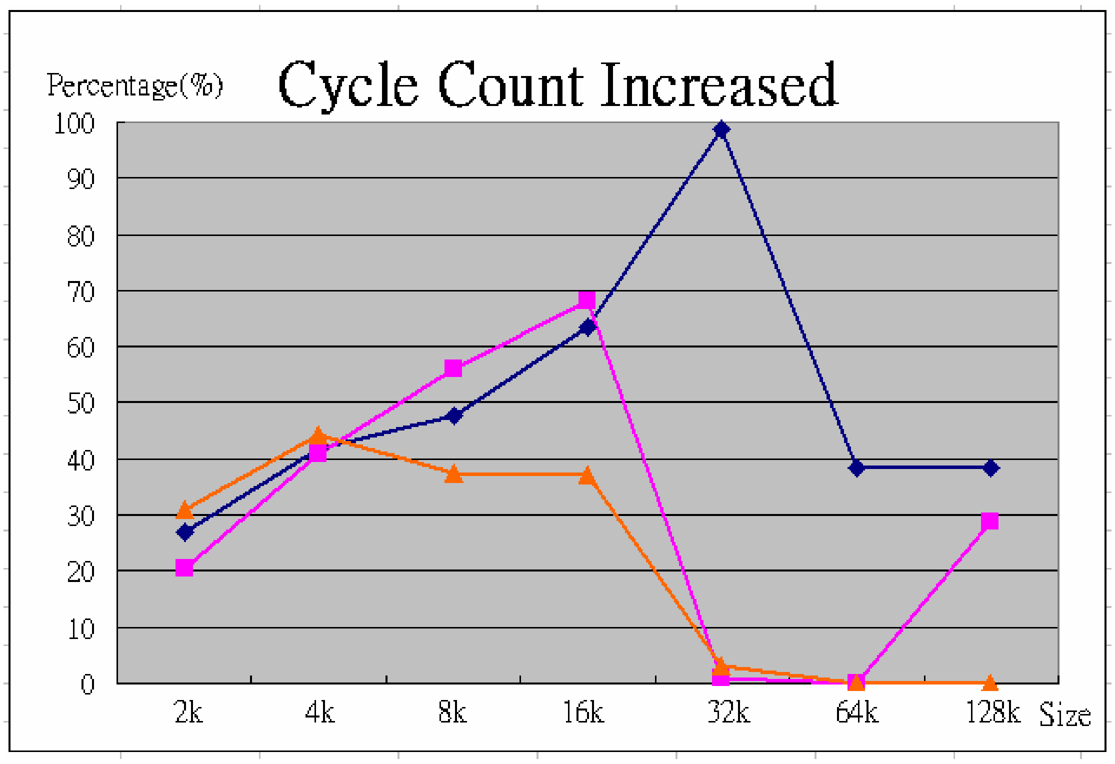


Figure 4.6: Cycle count increased on D-cache running JPEG encoder (***mem_short***)

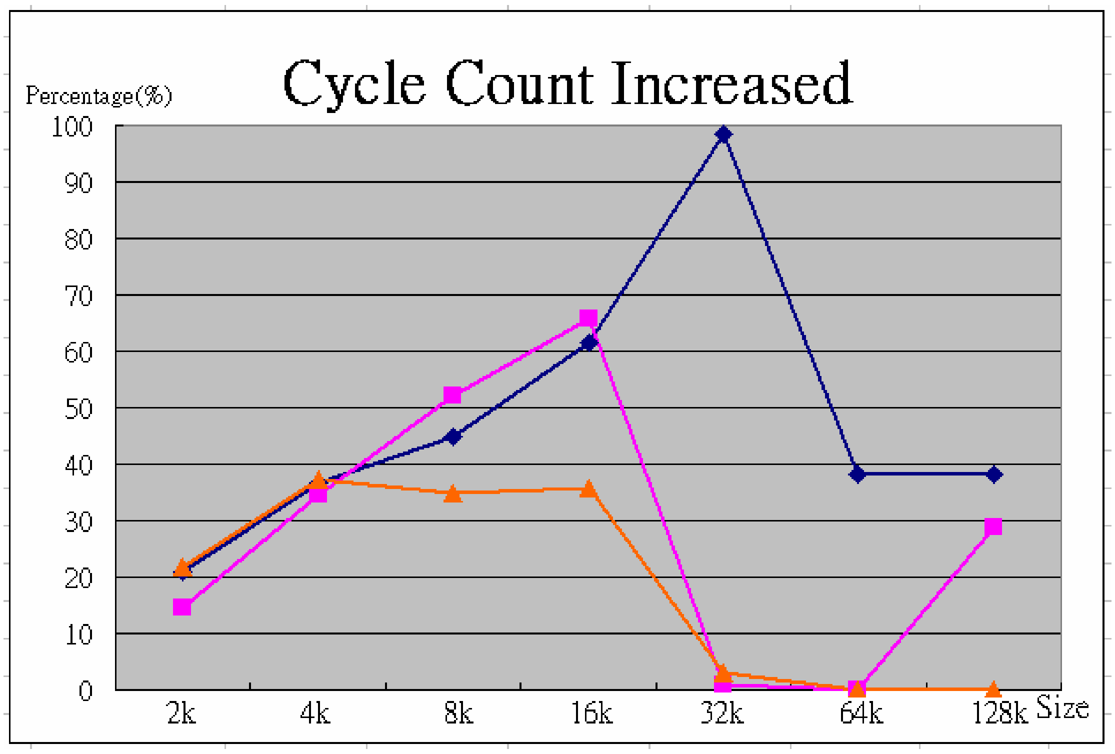


Figure 4.7 : Cycle count increased on D-cache running JPEG encoder (***mem_long***)

The cycle count on ***Seq 2 Way*** varies dramatically with respective to cache size. The cycle count on ***Seq 2 Way*** even doubles at the cache size of 32KB despite the fact that the miss rate is almost zero. Comparing to the cycle count on ***Seq+Pri*** of the same cache size, which is almost identical to conventional set-associative cache, it is clearly that the most of the frequently accessed blocks in ***Seq 2 Way*** were placed in the later accessed way the first time they were introduced to the cache and become stuck there therefore causing such huge increase in cycle count. However, the cycle count on ***Seq+Pri*** is still not stable enough with respective to miss rate due to lack of mechanism to promote recently accessed block to earlier accessed ways during cache hit. ***Seq+Pri+Pmt*** further increases the hit rate on the earliest accessed way by allowing the recently accessed blocks to be swapped to the earliest accessed way and further reduces and stabilizes the cycle count.

Comparing figure 4.6 with figure 4.7, the curve of the percentage of increased cycle count on ***mem_short*** actually shift down compared to ***mem_long*** on smaller cache sizes. The reason can be explained by considering the formula of cache access cycle count expressed as follows:

$$\text{Cycle\_count} = (\#\_\text{of\_access}) \times (\text{hit\_rate} \times \text{hit\_time}_{\text{cycle}} + \text{miss\_rate} \times \text{miss\_penalty}) \quad (4)$$

The overall cache access cycle count is composed of the total cycle count on cache hits and the total cycle count on cache misses. The main memory access latency affects the cache miss penalty therefore plays an important role in overall cache access cycle count. The longer main memory access latency increases the effect of the miss penalty on the overall cache access cycle count and therefore reduces the effect of the increased hit cycle count due to sequential way access on the overall cache access cycle count, especially when the miss rate is high. As the gap between main memory access time and processor clock rate continues to grow, the effect of the increased hit cycle time due to sequential way access on the overall cache access cycle count should continue to reduce in the future.

Table 4.10 and table 4.11 shows the cycle counts of each benchmark on 32-KB D-caches of different cache configurations with respect to ***mem_short*** and ***mem_long***. Sequential way access increases the average cache access cycle count dramatically while priority replacement policy and promotion placement policy effectively alleviates the increased cycle count by increasing the hit rate on the earliest accessed way. The average cycle count on ***Seq+Pri+Pmt*** increases only about 6.39% comparing to conventional set-associative cache. By comparing table 4.10 with table

4.11, it also shows that the overhead on the increase cycle count reduces as the cache miss penalty becomes more dominant.

32KB D-Cache (16-byte block, mem_short)							
	Cycle Count						Average Increment(%)
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	
2 Way Conv	6748001	363642	3082337	271533	198011	527029	—
Seq 2 Way	8577607	722289	4421579	278665	302000	855049	48.63
Seq+Pri	9692453	367415	4040671	278665	208743	853763	22.94
Seq+Pri+Pmt	7383729	374977	4006235	271857	217344	602263	7.34

Table 4.10: Cycle counts of benchmarks on 32-KB D-caches (***mem_short***)

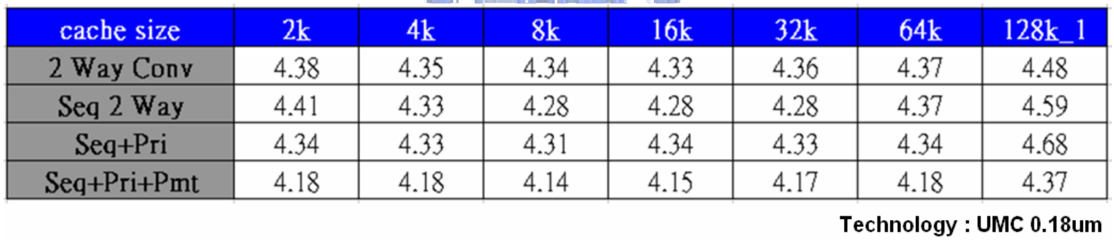
32KB D-Cache (16-byte block, mem_long)							
	Cycle Count						Average Increment(%)
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	
2 Way Conv	8029871	363912	5947467	271913	203361	528569	—
Seq 2 Way	9902527	722568	6566889	279045	307350	856639	41.35
Seq+Pri	10974323	367685	6185801	279045	214093	855303	18.57
Seq+Pri+Pmt	8665599	375247	6151365	272237	222694	603803	6.39

Table 4.11: Cycle counts of benchmarks on 32-KB D-caches (***mem_long***)

4.2.4 Cycle Time

The cycle time of each cache configurations for different cache sizes are shown in table 4.12. The cycle times are derived by synthesizing the RTL models of each cache configurations for different cache sizes using Artisan UMC 0.18um technology. Both tag array and data array are included in the model. Although sequential way access and smarter placement/replacement policies cause additional hardware overhead on the cache architecture, the cycle times of the proposed cache model do not increase dramatically compared to conventional set-associative cache of the same cache size. In fact, the cycle times of the proposed cache model actually reduce slightly.

Although proposed model increases the hardware complexity and therefore cause additional loading on the hit-signal, the increased loading is well compromised by the eliminated fanout on the hit signal used to select the hit data block in conventional set-associative cache discussed earlier. For (Seq+Pri+Pmt), since the latest accessed block is always placed to the earliest accessed way and since there is only two way, the need for LRU detection is no longer required because the blocks in the later accessed way are always the LRU blocks. By eliminating the LRU detection unit, the fanout on the hit-signal in (Seq+Pri+Pmt) is further reduced and consequently the cycle time of (Seq+Pri+Pmt) is also further reduced. The cycle time of (Seq+Pri+Pmt) is reduced by up to 4% in average compared to the conventional set-associative cache of the same cache size.



cache size	2k	4k	8k	16k	32k	64k	128k_1
2 Way Conv	4.38	4.35	4.34	4.33	4.36	4.37	4.48
Seq 2 Way	4.41	4.33	4.28	4.28	4.28	4.37	4.59
Seq+Pri	4.34	4.33	4.31	4.34	4.33	4.34	4.68
Seq+Pri+Pmt	4.18	4.18	4.14	4.15	4.17	4.18	4.37

Technology : UMC 0.18um

Table 4.12: Cycle time

4.2.5 Memory Access Time

The overall memory access time of each cache configuration can be derived by multiplying the overall cache access cycle count by the cache cycle time. The overall memory access time is the actual time the cache spent during simulation. Table 4.13 shows the overall memory access time of each benchmark on 32-KB D-caches of different cache configurations. Since the cycle time does not increase dramatically due to the reduced fanout on the hit-signal, the average percentages of increased memory access time on ***Seq 2 Way*** and ***Seq+Pri*** manage to be closed to their corresponding percentages of increased cycle count. For ***Seq+Pri+Pmt***, the smaller cycle time due to the elimination of the LRU detection unit further reduces its percentage of increased memory access time to only 1.96%.

32KB D-Cache (16-byte block, Long Latency)							
	Memory Access Time (ns)						
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	Average Increment(%)
2 Way Conv	34769341	1575739	25752532	1177383	880553	2288704	—
Seq 2 Way	42976967	3092591	28106285	1194313	1315458	3666415	40.01
Seq+Pri	46970102	1573692	26475228	1194313	916318	3660697	17.20
Seq+Pri+Pmt	35962236	1557275	25528165	1129784	924180	2505782	1.96

Table 4.13: Memory access time of benchmarks on 32-KB D-caches

4.2.6 Power-Delay Product

The power-delay product implies the overall energy consumption during the simulation. Table 4.14 shows the power-delay product of each benchmark on 32-KB D-caches of different cache configurations. The result shows that sequential way access together with priority replacement and promotion placement policies reduces the cache power consumption while preventing the overall memory access time from increasing dramatically, consequently has better energy efficiency compared to conventional set-associative cache. Up to 23.8% of reduction in power-delay product is achieved by ***Seq+Pri+Pmt*** compare to conventional set-associative cache.

32KB D-Cache (16-byte block, Long Latency)							
	Power-Delay Product						
Type	Sorting	Jpeg_enc	Jpeg_dec	Whet	FFT	Matrix	Average Reduction(%)
2 Way Conv	507903586	20023703	301711516	14738013	15239293	44522383	—
Seq 2 Way	511933038	34080972	275899725	10040228	17128579	52264745	-10.06
Seq+Pri	571616753	13219326	362493531	10705460	11190442	54170992	5.59
Seq+Pri+Pmt	380318625	12429858	330842462	9345908	11249490	32681417	23.79

Table 4.14: Power-delay product of benchmarks on 32-KB D-caches

Chapter 5 Conclusions

The thesis examines the concept of using sequential way access to reduce cache power consumption by reducing the number of ways being activated on each cache access. By accessing each way sequentially on each cache access and eliminating subsequent way accesses when a cache hit is detected, substantial energy can be saved at the expense of increased overall cycle count. In order to increase performance, smarter placement and replacement policies are applied to increase the hit rate on the earliest accessed way by placing the recently accessed blocks in the earliest accessed way. The increased hit rate on the earliest accessed way further reduces the number of ways being activated and the additional access cycles on each cache access, consequently further reduces both the overall cache power and the average cache access cycle count. Sequential way access also provides means to reduce the fanout on the hit-signal therefore prevents the cache cycle time from increasing due to more complex logic. By applying sequential way access together with priority replacement policy and promotion placement policy, experimental result shows that a 32KB 2-way sequential way-access set-associative cache reduces the power-delay product by an average of 23.8% compared to the conventional 2-way set-associative cache of the same size.

References

[1] P. Gelsiniger. Microprocessors for the New Millennium – Challenges, Opportunities and New Frontiers. In proceeding of the *Int'l Solid-State Circuits Conference*, Feb. 2001.

[2] S. Santhanam, "Strongarm sa110 –a 160mhz 32b 0.5w cmos arm processor-," In proceeding of the *Hot Chips 8: A Symposium on High-Performance Chips*, Aug. 1996.

[3] N. P. Jouppi, P. Boyle, J. Dion, M. J. Doherty, A. Eustace, R. W. Haddad, R. Mayo, S. Menon, L. M. Monier, D. Stark, S. Turrini, J. L. Yang, W. R. Hamburgen, J. S. Fitch, and R. Kao, "A 300-mhz 115-w 32-b bipolar ecl microprocessor," In proceeding of the *IEEE Journal of Solid-State Circuits*, volume 28, pp. 1152-1166, Nov. 1993.

[4] J. H. Edmondson and et al. Internal organization of the Alpha 21164, a 300-MHz 64-bit quad-issue CMOS RISC microprocessor. *Digital technical Journal*, 7(1), 1995.

[5] A. Hasegawa et al, "SH3: High Code Density, Low Power," In proceeding of the IEEE *Micro*, pp. 11-19, Dec. 1995.

[6] B. Calder, D. Grunwaldm, and J. Emer, "Predictive Sequential Associative Cache," In proceeding of the Int'l Synposium on High Performance Computer Architecture, pp. 224-253, Feb. 1996.

[7] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," In proceeding of the 1999 International Symposium on Low Power Design, pp. 273-275, Aug. 1999.

[8] M. Powell, A. Agarwal, T. Vijaykumar, B Falsafi, K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," In proceeding of the 32th International Symposium on Microarchitecture, Austin, TX, USA, pp. 54-65, Dec. 2001,

[9] T. Johnson and W.W. Whu, "Run-time Adaptive Cache Hierarchy Management via Reference Analysis," In proceeding of the *OSCA-24*, Jun. 1997.

[10] M. Huang, J. Renau, S.M. Yoo, and J. Torrellas, "L1 Data Cache Decomposition for Energy Efficiency," In proceeding of the Low Power Electronics and Design, International Symposium, pp. 10-15, 6-7 Aug. 2001.

[11] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip cache," In proceeding of the Ninth international Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS X), pp. 211-222, Oct. 2002

[12] Zeshan Chishti, Michael D. Powell, and T. N. Vijaykumar, "Distance Associativity for High-Performance Energy-Efficient Non-Uniform Cache Architectures" In proceeding

of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 55-66, 2003.

[13] G. Reinman and B. Calder, “Using a serial cache for energy efficient instruction fetching,” In proceeding of the Journal of System Architecture, volume 50, issue 11, pp. 675-685, Nov. 2004.

[14] S.Y. Cheng and J.D, Huang, “Low-Power Instruction Cache Architecture Using Pre-Tag Checking,” IEEE int’l Symp. In proceeding of the VLSI Design, Automation, and Test, pp. 83-86, Apr. 2007.

[15] G. Tyson, M. Farrens, J. Matthews, and A.R. Pleszkun “A Modified Approach to Data Cache Management,” In proceeding of the 28th Annual International Symposium on Microarchitecture, pp. 93-103, Dec. 1995.

[16] J.A. Rivers and E.S. Davidson “Reducing Conflicts on Direct-Mapped Caches with a Temporality-Based Design,” In proceedings of the 1996 International Conference on Parallel Processing, pp. 151-162, Aug. 1996.

[17] V. Mulutinovic, B. Markovic, M, Tomasevic, and M. Tremblay, “The Split Temporal.Spatial Cache: Initial Performance Analysis,” In proceedings of the SCIzzL-5, Santa Clara, California, USA, Mar. 1996.

[18] J.Sahuquillo and A. Pont, “The Filter Cache: A Run-Time Cache Management Approach,” In proceeding of the 25th EUROMICRO conference, pp. 424-431, Sep. 1999.