12-12-2014

Improving Performance and Energy Efficiency of On-Chip Memory Systems

Menglong Guan
menglong.guan@uconn.edu

Improving Performance and Energy Efficiency of On-Chip Memory Systems

Menglong Guan

B.Eng. in Electronic Engineering,
Nanjing University of Science and Technology, China, 2012

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

At the

University of Connecticut

2014

APPROVAL PAGE

Master of Science Thesis

Improving Performance and Energy Efficiency of On-Chip Memory Systems

Presented By

Menglong Guan, B.Eng.

Major Advisor _____

Lei Wang

Associate Advisor _____

John Chandy

Associate Advisor _____

Helena Silva

University of Connecticut

2014

Acknowledgements

First and foremost I would like to express my sincerest gratitude to my advisor, Dr. Lei Wang, who assisted me throughout the whole M.S. degree studying especially during the thesis preparation with his patience. His encouragement and education are the most important power to help me overcome all hardness. I would also like to thank my associated advisors, Dr. John Chandy and Dr. Helena Silva for all supports for my thesis writing.

Also, I would like to bring my gratitude to my parents, Ming Guan and Aiping Meng, who not only financially support me to have the chance studying at University of Connecticut, but also unconditional care me thought all my life.

In my daily work and study, I also received uncountable number of assistance from my knowledgeable fellow labmates and friends, Junlin Chen, Guoxian Huang, Fengyu Qian, Wenjie Huang, Ridvan Umaz, Shuai Chen, Dong Zhao, Qingchuan Shi. I would like to express my thankfulness to all of them, since I was not able to so smoothly finish my M.S. study without their help.

What's more, I would also like to appreciate all happiness and cheerfulness experiences shared with my friends, Ting Feng, Weiqiang Chen, Huizhong Gao, Zhitong Fei, Ran Miao, etc. It was them who colored my life with warm instead of lonely.

In all, my degree would not have been possible without the encouragement and assistance of all the aforementioned. I would also like to thank some missed ones who have also given me innumerable supports. I am confident that I will miss all of you in my future work and life. Thank you all!

Contents

List of Tables

List of Figures

Improving Performance and Energy Efficiency of On-Chip Memory Systems

Menglong Guan, M.S.

University of Connecticut, 2014

ABSTRACT

Better performance and lower power consumption are never ending topics for computer systems or embedded systems, especially for today's "carry on" computers or smart phones. The memory system, including cache or main memory, not only requires very high read and write speed, but also low power consumption to keep transistors in a good thermal condition. Current mobile devices have been using 3D DRAM systems for their larger capacity and better performance. Three-dimensional (3D) integration allows IC designs to stack DRAM directly on the top of execution units, which greatly reduces DRAM access latency and optimizes energy consumption. Unfortunately, the heat generated by the processor unit cannot be effectively dissipated. As a result, DRAM operation temperature is undesirably increased. Due to the fact that 3D-stacked DRAM operates under severe thermal conditions, conventional designs based on peak temperatures lead to a high refresh rate, which introduces a large performance penalty in 3D-stacked DRAM. To address this problem, a *Temperature Aware Refresh* (TAR) technique for 3D-stacked DRAM is proposed in this thesis. The goal is to mitigate the performance penalty by adjusting the refresh rates of DRAM banks based

on the actual thermal conditions at their locations. As a result, only banks that reach their peak temperature refresh frequently, and the rest of banks can be refreshed at a lower rate. This enables more read and write accesses, which improves the overall system performance. Simulation results show that more than 10% improvement over the all-bank refresh and 3 to 9% improvement over the per-bank refresh have been achieved.

Although studies have shown that the 3D stacked DRAM has the potential to be used as cache memory, the conventional SRAM cache structure still dominates the current embedded processor cache memory system. Recent research has improved the performance and saved the energy consumption of cache memory. We propose a new cache design technique, referred to as *Early Tag Access* (ETA) cache, to improve the energy efficiency of data caches in embedded processors. The proposed technique performs early tag accesses to determine the destination ways of memory instructions before the actual cache accesses. ETA techniques thus enables only the destination way to be accessed if a hit occurs during the early tag access. The proposed ETA cache can be configured under two operation modes to exploit the tradeoffs between energy efficiency and performance. It was shown that our technique is very effective in reducing the number of ways accessed during cache accesses. This enables significant energy reduction with negligible performance overheads. Simulation results demonstrate that the proposed ETA cache achieves over 52.8% energy savings on average in a L1 data cache and a Translation-Lookaside-Buffer (TLB).

Chapter 1

Temperature Aware Refresh for DRAM Performance Improvement in 3D ICs

1.1 Introduction

3D ICs employ multiple dies stacked on top of each other and connected by through-silicon vias (TSVs). Compared with 2D ICs, interconnect length is reduced, thereby enabling significant delay reduction. It was shown that vertical signal latency of 20 layers is only 12 ps [1]. However, since power density becomes worse, 3D ICs suffer from the increased peak temperature. Furthermore, as thinner wafers with short TSVs are utilized to lower TSV resistance and allow more wafers to be stacked in the same volume, the power density is further increased, which causes higher peak temperatures. Existing work [2] reported hot spots of 88.35°C, and in some extreme situations, the peak temperature can be over 100°C [1].

One trend in 3D IC design is to stack processor and memory dies on top of each other. In particular, 3D-stacked DRAM can reduce memory access latency and increase memory access bandwidth. It also helps to reduce the overall power consumption [2], [3], [4]. It was also shown that using 3D-stacked DRAM as L2 cache has better performance than 2D SRAM L2 cache [4].

One option for 3D memory is to stack several memory dies which are designed in conventional 2D SRAM or DRAM ways and connected by TSVs [2]. Another approach is the "true" 3D DRAMs [5] which have individual bitcell arrays stacked in the 3D structure. Although there are a lot of 3D-stacked structure designs, the topology of these processor-memory 3D organization keeps the same with conventional off-chip DRAMs.

Using 3D-stacked DRAM has the benefits of high packaging density and wide memory bandwidth as compared with conventional 2D DRAM memory. However, this technique comes with some new challenges. First, the heat generated by the microprocessor cannot be

well dissipated due to different layers blocking the path from the processor to the heat sink. As a result, hot spots with high peak temperature are created in the 3D-stacked DRAM structure [2], [6]. Second, processors in a multi-core system have various workloads so that power consumptions differ widely. The DRAM banks on the top of execution units are usually much hotter than those on top of the private L2 cache, with a thermal variance of at least 30°C. Although not all parts of 3D-stacked DRAM operate in the peak temperature, the configuration of DRAM (e.g., refresh rate) is still defined by the peak temperature.

DRAM devices require periodical refresh operations to combat leakage currents that drain charges from DRAM cells. The refresh operation is temperature-dependent, e.g, every 64ms or 32ms at above 85°C as specified by the DRAM standards [7]. The initial refresh operation, Burst Refresh, continuously refreshes all DRAM rows, but this mode requires a long period of time within which the DRAM cannot process read or write requests. The Distributed Refresh mode, which is supported by the JEDEC standard, tries to mitigate this issue. In this mode, all rows in a DRAM bank are divided into 8K groups, which is because memory banks initially had 8K rows. The time interval between two different refresh signals is 7.8μs (3.9μs at above 85°C), which is denoted as *Refresh Interval* or tREFI.

As the capacity of DRAM increases, the number of rows also increases. Initially, memory banks had 8K rows, so each refresh signal refreshes exactly one row. Though the capacity of DRAM devices increased a lot, the 8K groups design has never changed, thereby within the same refresh interval the number of rows increased. Consequently, the time period of each refresh operation for one group increased as well, thus the overall time for each refresh operation, defined as *Refresh Cycle Time* or tRFC. DRAM chip density doubles every two to three years (projected to 32G by 2020 [8]). The tRFC grows while tREFI shrinks as 3D-stacked DRAM works in higher temperatures. As a result, the available time for read and write operations decreases. This is expected to be a problem that significantly affects the performance of 3D-stacked DRAM.

Conventional refresh operations for all banks are required at the same period of time,

and determined by the operating temperature. This is because conventional DRAM devices use peak temperature to determine the refresh rate. A modern DRAM system consists of multiple ranks, and each rank is formed by multiple banks. Different ranks or banks can be individually accessed. There are two schemes for refresh operations in DRAM: *all-bank refresh* and *per-bank refresh*.

In conventional 2D off-chip DRAM design, temperature for all banks almost keep the same. However, hot spots in 3D-stacked DRAM have a large temperature gradient, which can be as high as 57°C [10]. In addition, the vertical direction conducts heat better than the horizontal direction [1], so that the overall design arrangement also impacts thermal conditions for different banks [2], [6]. Thus, in both *all-bank refresh* and *per-bank refresh* schemes, many banks actually have much lower temperatures than the peak temperature. These banks do not need to be refreshed at the rate determined by the peak temperature.

This chapter proposes a new *Temperature Aware Refresh* (TAR) technique to address the aforementioned problem. The basic idea is to keep track of each bank's temperature so that the bank refresh rate or refresh period can be adjusted independently for each bank based on its actual thermal conditions. Simulation results demonstrate that the proposed technique can significantly improve the performance of 3D-stacked DRAM.

1.2 Related Work

A lot of work [11]–[17] aimed at optimizing the refresh operation in DRAM as it greatly affects the overall memory performance. Stuecheli *et al.* proposed Elastic Refresh [11], where the refresh operation can be delayed or postponed up to eight refresh commands when conflicting with performance-related operations, such as DRAM read and write. Kim *et al.* proposed Subarray-level Parallelism [12] that enables multiple accesses to the same bank. A bank is divided into several subarrays so that they are operated independently and have their own row-buffers. Nair *et al.* proposed Refresh Pausing [13], which pauses the

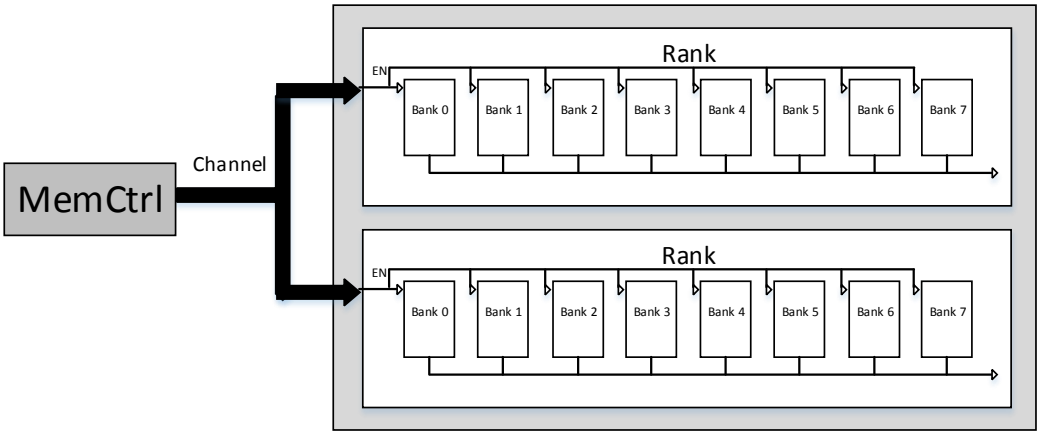
on-going refresh operation to issue other more important pending memory requests. Ghosh *et al.* proposed Smart Refresh [14], which decreases the number of refresh commands based on the live retention time of rows. Liu *et al.* proposed Retention-Aware Intelligent DRAM Refresh [15] to intelligently group multiple rows into bins based on their required refresh rates, which are determined in the manufactory process. Kalyan *et al.* proposed Scattered Refresh [16] to distribute rows to several subarrays so that subarrays can be refreshed in a pipelined manner. Chang *et al.* proposed Parallelizing Refresh [17], which improves the degree of parallelism at the bank level or subarray level.

Note that most of these existing work adjust all the banks or subarrays in DRAM with the same refreshing rate to mitigate the performance loss due to refreshing. While they are effective in 2D DRAM, high temperature variations in 3D-stacked DRAM will incur large performance degradation if the refresh rate determined only by the peak temperature is used. In this chapter, we propose the Temperature Aware Refresh (TAR) technique to adjust the refresh rate by considering the impact of temperature variations caused by the stacked structure of 3D DRAM. It is worth mentioning that the proposed technique is compatible with the existing DRAM optimization techniques, i.e., it can be applied in combination with these techniques to 3D DRAM.

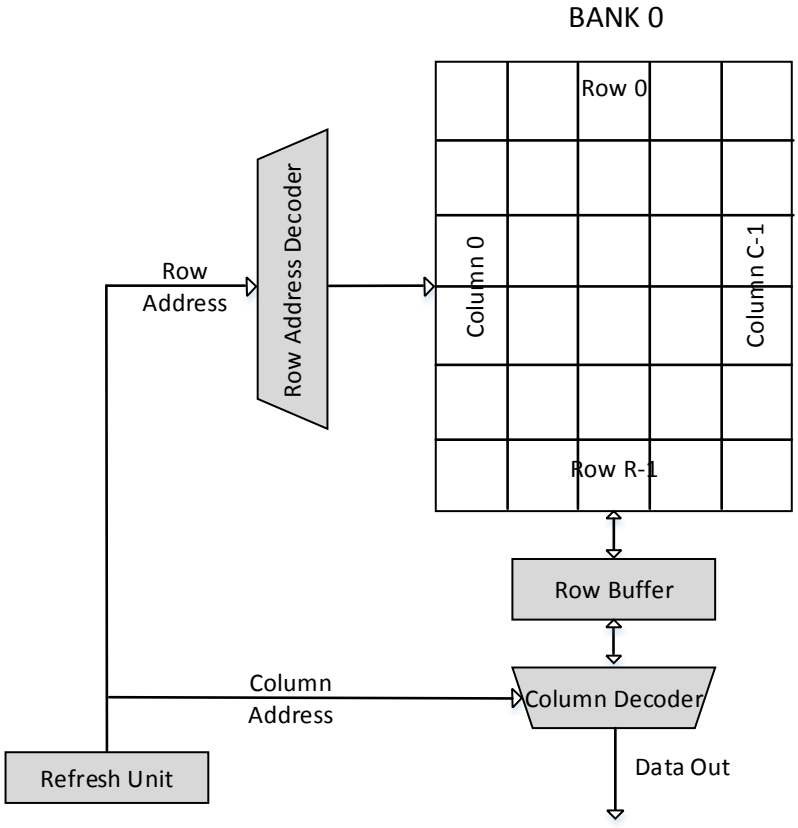
1.3 Background

1.3.1 DRAM and 3D Structure

Fig. 1.1 shows the baseline DRAM structures. DRAM based memory system are consist of channels, ranks, banks, rows and columns. Each channel controlled by own memory controller is a set of ranks that share a common address and data bus with processor. A rank is formed with several banks that share an inner data bus. Different banks can be requested in parallel, proving bank-level parallelism. Columns consists a row, and only one



(a) The structure of one channel



(b) The structure of one bank

Figure 1.1: The structure of DRAM

row in bank can be accessed at the same time.

By stacking processor or memory dies on top of each other, 3D-stacked DRAM is a solution for reducing memory access latency and increasing memory access bandwidth between the last-level on-chip cache and 3D-stacked DRAM. Different memory stacked options also has various performance improvement. Fig. 1.2 shows one option for 3D memory that stacks several memory dies which are designed in conventional 2D SRAM or DRAM ways, and connected by TSVs [2].

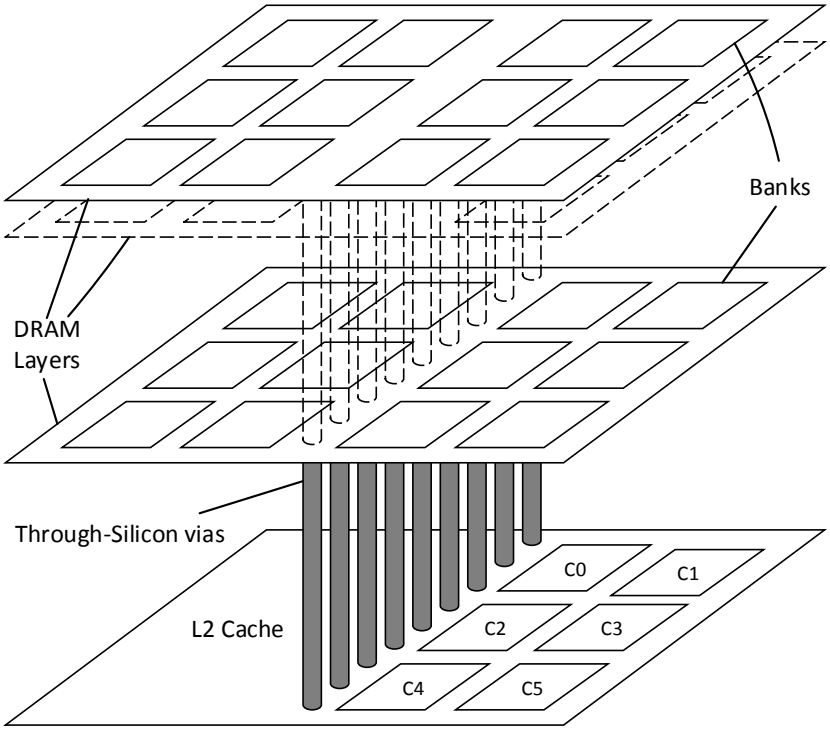


Figure 1.2: 3D-Stacked DRAM with conventional 2D DRAM design

1.3.2 All Bank Refresh and Per Bank Refresh

In *all-bank refresh*, DRAM chips refresh all banks in a pipelined order, and every refresh signal concurrently refreshes a group of rows in every bank, which prevents all banks from

being accessed until the refresh operation is complete. The reason for using pipelined bank refresh instead of simultaneously refresh is that it is unsustainable for the power delivery network to do simultaneously refresh [9]. The time period for completing this all bank refresh operation is named tRFCab which is a function of DRAM device's capacity. *All-bank refresh* is applied to both commodity double data rate (DDR) and low power DDR (LPDDR) DRAM. As shown in Fig. 1.3, the second READ operation of bank 0 and Write operation of bank 1 have to wait the REF complete.

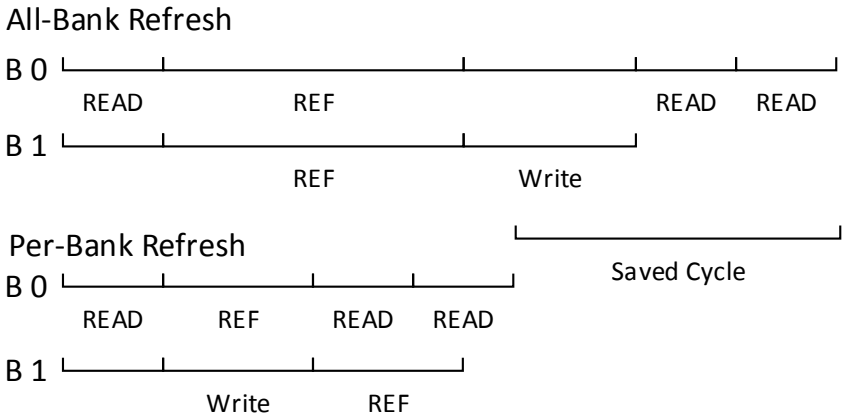


Figure 1.3: All-bank refresh and Per-bank refresh

In order to reduce the unavailable time of refreshing, mobile platform LPDDR DRAM uses a partial accessible refresh scheme *per-bank refresh*. *per-bank refresh* divides a refresh operation into non-overlapped per-bank refresh operations. Banks that are not refreshing are able to process other memory requests. As a result, a REFpb is created by memory controller N_{banks} times frequently than a REFab, which is tREFIpb = tREFIab/N_{banks}. The reason for the decision, using non-overlapping per-bank refresh instead of overlapped one, is to avoid new timing constraints, which is made by the LPDDR DRAM standard committee [24]. In addition to non-overlapping refresh, every bank in the rank accomplishes refresh command in a strict sequential round-robin order. Since this scheme only blocks the bank

that is refreshing, the overall performance penalty due to refresh is mitigated. Fig. 1.3 shows the Saved Cycle. Bank 1 does the WRITE while the Bank 0 is refreshing, and Bank 0 does the second READ when Bank 1 is refreshing.

1.3.3 Memory Controller and Refresh Operation

Fig. 1.4 shows the refresh operation flow generated by Memory Controller. As CPUs sent read and write operations to memory controller. These operations are stored in Input Queues in a first in first out manner, and then sent into the proper Rank Queue or Bank Queue based on devices configuration. While the tREFI counter for the rank expires, a refresh signal created by memory controller stored into Refresh Queue until it is sent to the DRAM device. In case of data losing the refresh command has a high priority than read or write, so when the rank completes current operation refresh command is forwarded into rank before other request. As a result, the refresh commands postpone read and write requests.

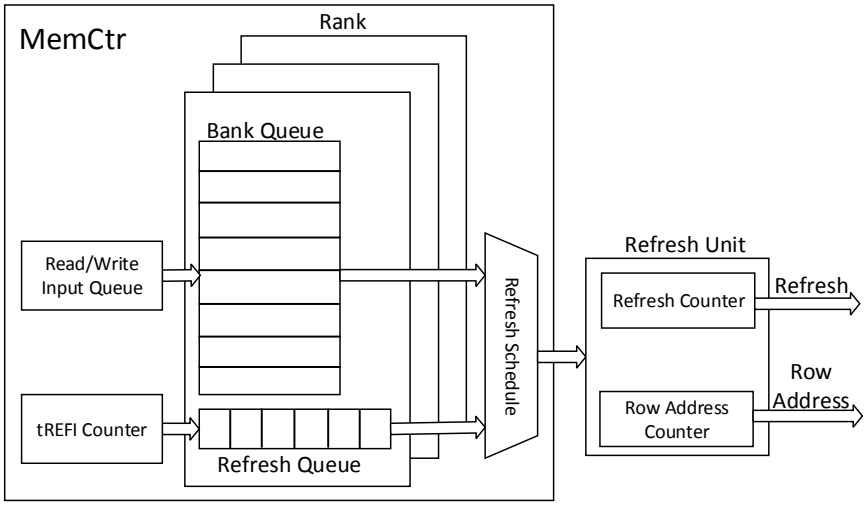


Figure 1.4: Memory Controller for refresh operation

1.4 Temperature Aware Refresh (TAR)

1.4.1 Motivation

The per bank refresh policy has mitigated some of the refresh penalty, however, both policies still suffer from hot spot problems that degrade the DRAM performance in 3D structures, since the conventional refresh rate is determined by the shortest retention time of DRAM cell. Refresh commands usually have a higher priority than access commands, such as read and write, to ensure data correctness. When a DRAM conducts a refresh operation, other commands will be blocked, which will cause read or write delays and eventually degrade the overall system performance. This performance loss cannot be neglected especially for DRAM with large capacity. It was reported [11], [17] that the estimated tRFC of 32G DRAM will be larger than 600ns. [16] shows the present tRFC latency number of 16Gb and 32Gb. The solid line presents the estimation calculated from 1, 2, and 4Gb device, and the dotted line presents the trend calculated from 4 and 8Gb device. Since the estimation from 4, 8Gb is according to the new DDR standard, we use this estimation as of simulation configuration.

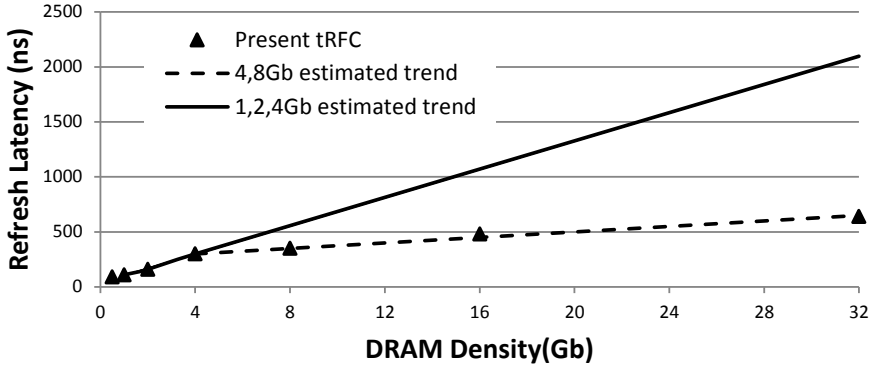


Figure 1.5: Refresh Latency tRFC trend

In conventional DRAM, the peak temperature determines the operating (e.g., refreshing) temperature [1]. The operating temperature will be over 100°C as power density increases

due to the higher frequency or more stacked dies. High temperature significantly reduces the retention time of DRAM cells, which requires a higher refresh rate. The JEDEC standard specifies the retention time for DRAM as 64ms for 85°C and lower, and 32ms for $85 - 95$°C. The refresh rate higher than 95°C is calculated by previous work [18], which claims 24ms for 100°C.

The Refresh Duty Cycle (RDC) is defined as the ratio of tRFC to tREFI. RDC indicates the percentage of the time that a DRAM rank is doing refresh within the refresh interval. The refresh rate for temperature higher than 95°C can be derived by [18], which assumes 16ms retention time for 105°C. The tRFC of 16G and 32G device can be obtained from the normal refresh mode of [16]. Table 1.1 shows that a 16G DRAM uses 12% of time to do refresh during the temperature range of $85 - 95$°C. In a worse case, a 32G DRAM working at the temperature higher than 105°C spends 60% of time in refresh operations.

Table 1.1: Refresh Duty Cycle for Different Temperatures

Temp (°C)	45	65	85	95	105	Hotter than 105
Data Retention Time (ms)	128	96	64	32	16	8
Ref Interval (μs)	15.63	11.72	7.81	3.91	1.95	0.98
Memory Density (tRFC)	Refresh Duty Cycle (tRFC/Ref Interval)					
1G (110ns)	0.70%	0.94%	1.41%	2.82%	5.63%	11.26%
2G (160ns)	1.02%	1.37%	2.05%	4.10%	8.19%	16.38%
4G (300ns)	1.92%	2.56%	3.84%	7.68%	15.36%	30.72%
8G (350ns)	2.24%	2.99%	4.48%	8.96%	17.92%	35.84%
16G (480ns)	3.07%	4.10%	6.14%	12.29%	24.58%	49.15%
32G (640ns)	4.10%	5.46%	8.19%	16.38%	32.77%	65.54%

Note that in conventional DRAM, the maximum temperature gradient can reach 57°C [10]. Although 3D structures have improved the performance by mitigating the wire latency and providing wide bandwidth, the DRAM refresh operation determined by the peak temperature will introduce large performance penalty. As a result, most read or write requests are

postponed to wait for refresh to be completed. Our goal in this chapter is to develop a technique that allows 3D-stacked DRAM to refresh at different rates in accordance with the actual temperature variations.

1.4.2 The Proposed Technique

The limitation of the conventional all-bank and per-bank refresh schemes is that the same refresh rate is applied to all banks. Per-bank scheme strictly refreshes banks in a sequential round-robin manner at the same refresh rate, even though it allows non-refresh banks to be accessed. When these refresh schemes are utilized in the 3D-stacked DRAM, all banks will be forced to operate at a high refresh rate determined by the peak temperature, e.g., from the bank on the top of the execution unit. This affects how DRAM can process read and write requests.

The reason is DRAM devices have limited bank refresh information in memory controller. The refresh unites determine which the next refresh bank is and which the next refresh row is. To address this problem, the bank selection logic should be moved into memory controller which is responsible for not only the time control but also the refresh bank selection. As a result, memory controller can signal each bank separately. The refresh units keep the function of selecting which the next group or row is for next refresh signal, but the bank ID information is given by memory controller. Memory controller stores varies refresh rate for different banks according to banks' operation temperature.

Fig. 1.6 shows a DRAM employing the proposed Temperature Aware Refresh (TAR) technique. Here we introduce several new components in each rank, including $(N_{banks} - 1)$ tREFI counters, a bank decoder in the memory controller, $(N_{banks} - 1)$ row address counters in the refresh bank control logic and $2 \times N_{banks}$ bits in the mode register which is a part of DRAM control unit. The detailed design of these components will be discussed later.

The proposed TAR technique will reduce the refresh rate for banks that work at the low

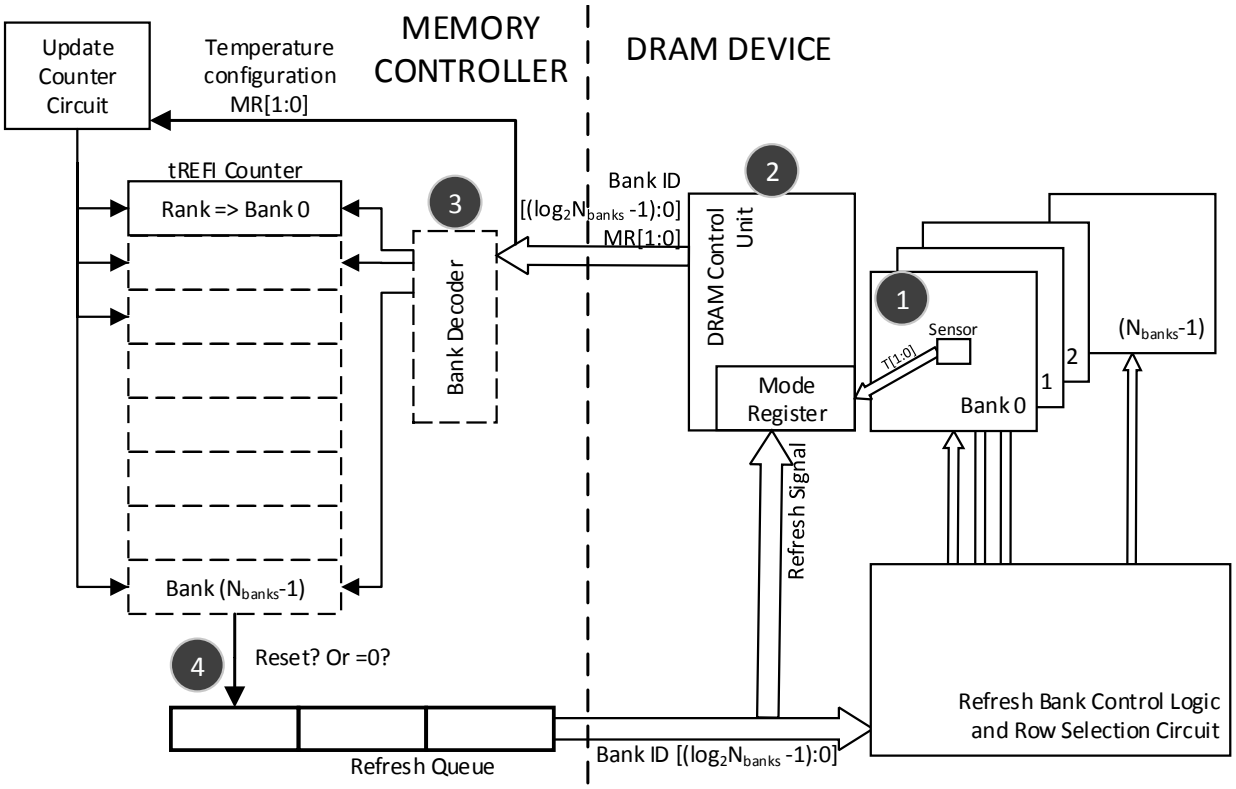


Figure 1.6: The proposed of Temperature Aware Refresh (TAR) technique.

temperature, thereby increasing the available time for read and write operations. Under the conventional refresh operation, the refresh rate for the entire rank is controlled by the memory controller, which has only one tREFI counter to control the refresh rate. There are also two different modes in the tREFI counter: the memory controller can adjust DRAM refresh operation at either the normal rate for $0 - 85°C$ or the high rate for $85 - 95°C$. If a bank works in the high temperature range, the memory controller will use the high refresh rate mode. This implies a shortcoming that instead of controlling the refresh rate of each bank, the memory controller only implement the same refresh rate for whole rank.

The proposed TAR technique keeps different refresh rates for different banks in the memory controller by adding additional tREFI counters. A bank decoder is introduced to select the tREFI counter. As a result, the new memory controller not only controls the refresh rate but also selects the bank, so that it can refresh each bank independently. Temperature sensors in the conventional DRAM report the temperature range of each bank. The DRAM control unit collects this information and informs the memory controller for refresh rate adjustment.

The detailed operation of the proposed TAR technique is illustrated in Fig. 1.6. First, the sensor in a given bank (bank 0 in this example) detects the temperature range and changes the corresponding bits of the mode register to the new temperature setting. The mode register, which is a part of DRAM control unit, is used to store various DRAM settings. The DRAM control unit is responsible for all the operations, such as read/write, and refresh requests. Second, the temperature setting and the ID of the bank, which is represented by the index of the temperature setting bits, will be sent to the memory controller. Third, the temperature setting will require the update counter circuit to update the corresponding tREFI counter with the new refresh rate. According to the ID of the bank received from the DRAM control unit, the bank decoder enables the associated tREFI counter so that it is reset by the update counter circuit. Since each bank is working at the different temperature, the tREFI counter of each bank will have a different number that represents the different

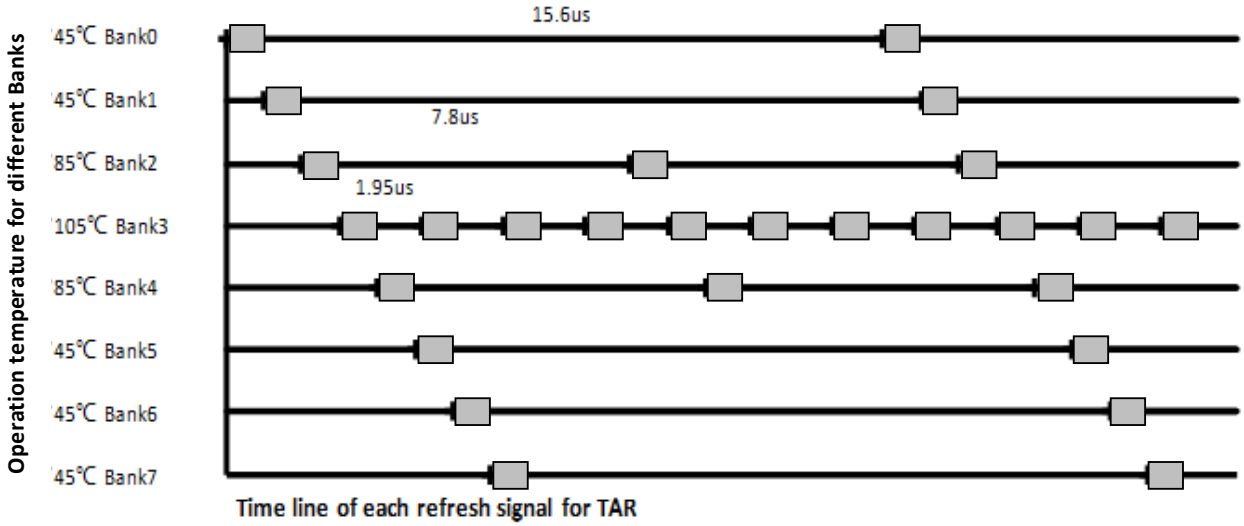
refresh rate. Finally, all tREFI counters will decrease by one in every clock cycle. When one of the tREFI counters expires, the refresh signal along with the ID of the bank will be sent to the refresh queue and then forwarded to the DRAM control unit. Consequently, the memory controller will allow more frequent refresh operations only for banks with the high temperature, while those with low temperature will be refreshed at a lower rate.

Let's consider the example shown in Fig. 1.7a. Assume that bank 3 is on the top of CPU main execution unit, which results in reaching the peak temperature $95 - 105°C$. The adjacent banks 2 and 4 have the operation temperature less than 85°C, and the rest banks are on the top of L2 cache that has the lowest temperature. The memory controller only sends refresh signals at the marked times (e.g., 1.95μs, 7.9μs, and 15.9μs for different banks). The duration between two timing marks is tREFI and the gray blocks represent the tRFC consumed by the refresh operations. The rest of the time period is available for read or write operations. The reason for these banks not starting at the same time is to avoid multiple refresh requests in the same cycle. In Fig. 1.7b, the conventional per-bank refresh scheme starts in different times as well, but all the banks will work at the same refresh rate as bank 3. On the other hand, the conventional all-bank refresh scheme in Fig. 1.7c starts from the same time point, but all banks will be unavailable during tRFC, which is 2.3 times longer than that of the per-bank refresh scheme [25]. Obviously, the proposed TAR technique releases more time for read and write operations in those banks with lower operation temperature.

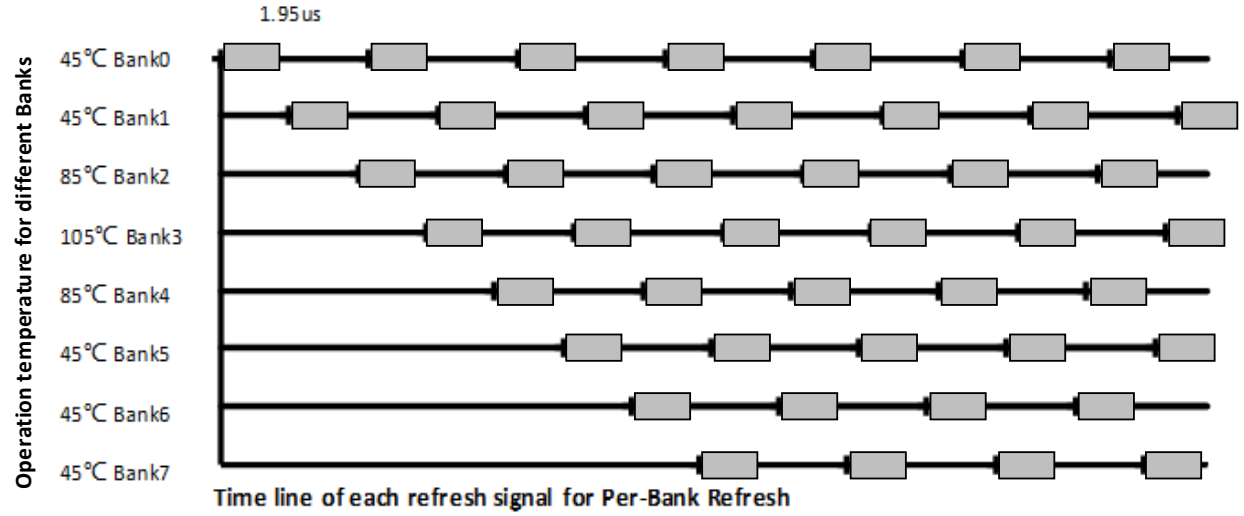
1.4.2.1 Temperature Sensor and Mode Register

We now discuss the hardware support for the proposed TAR technique. There are three important factor for temperature sensor to achieve our proposed technique: the auto switch function, the location of temperature sensor, and the temperature range .

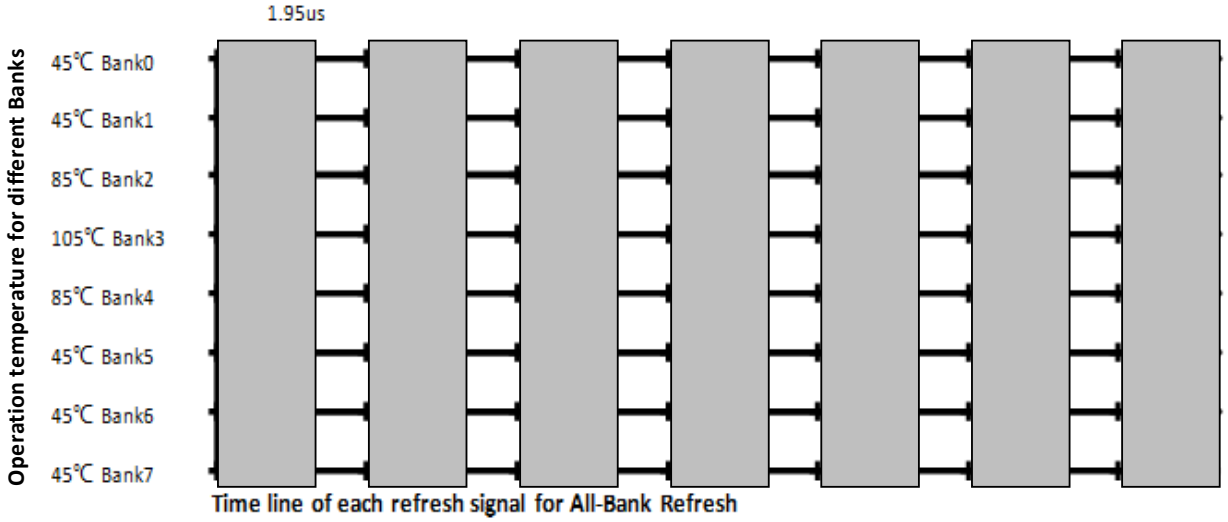
In conventional DRAM chips, each bank has a temperature sensor, which can detect the operation temperatures in DRAM banks and enable two modes of refreshing: normal mode for temperature lower than 85°C and extended mode for temperature $85 - 95°C$ [19].



(a) An example of temperature aware refresh operations



(b) An example of temperature aware refresh operations



(c) An example of temperature aware refresh operations

Figure 1.7: Examples of Three Different Refresh Policy

There are also two control modes, Self Refresh Temperature (SRT) and Auto Self Refresh (ASR), for different temperature ranges. SRT manually forces DRAM to refresh frequently, while ASR supports automatic switch from the low refresh rate to the high refresh rate when the operation temperature rises over 85°C. This indicates that temperature sensors can automatically detect the operation temperatures in DRAM banks.

Other considerations for temperature sensors are location and temperature range selection. It is known that DRAM does not have large temperature variations within the same bank [6], but the temperature differences are large between each function unit. For example, in a 4-core 2D processor the largest temperature difference is 30°C, which is between the floating point adder and the L2 cache [6]. Thus, the proposed TAR technique requires one temperature sensor located in the center of each bank.

For temperature range selection, it was shown [1] that the vertical direction dominates the main thermal conduction. Therefore, in 3D structures the banks on the top of the execution units of the logic layer will have larger temperature variances than the adjacent banks. Table 1.1 shows that the main performance impact of refresh operations is between $65 - 105$°C, where the RDC doubles from 45°C to 85°C and then doubles every 10°C over 85°C. Additionally, in the proposed TAR if one bank's tREFI counter is reset, the memory controller will send a refresh signal to this bank. For example, the tREFI counter for bank 0, whose tREFI = 3900ns, resets at 2000ns, so bank 0 will be refreshed 1900ns before the supposed refresh operation. Consequently, by increasing the number of temperature ranges the DRAM will refresh more frequently when the temperature crosses different temperature ranges. It is possible to have many temperature ranges; however, considering the hardware overhead, it is impractical to design a temperature sensor that has too many temperature thresholds. Our technique utilizes three temperature thresholds, i.e., 45°C, 85°C, 95°C. As a result, there are four temperature ranges for each bank, $0 - 45$°C, $45 - 85$°C, $85 - 95$°C, $95 - 105$°C. Temperature higher than 105°C is rare in 3D ICs and thus is not considered here (althougt it is easy to extend the proposed technique to this range). Previous work has

shown that many temperature sensors [20] [21] are able to detect these temperature ranges.

Table 1.2: Range Select Logic and Mode register settings

Operation Temperature	Input	Output	
	Sensor Register	MR[1]	MR[0]
0°C to 44°C	000000 to 0101100	0	0
45°C to 84°C	0101101 to 1010100	0	1
85°C to 94°C	1010101 to 1011110	1	0
95°C to 105°C	1011111 to 1101001	1	1

In conventional DRAM chips, for the entire rank there is one bit in the mode register that controls the refresh rate. If one of the banks is working at the high temperature, the bit will flip into the high temperature mode so that the memory controller will refresh this rank at the high rate. In the proposed TAR, every bank in a rank has its own temperature range. Thus, there are $2 \times N_{banks}$ bits in total required in the mode register for each rank. The third column of Table 1.2 shows TAR requires 2 bits in the mode register to represent 4 temperature ranges. The temperature sensor shown in [21] has the temperature resolution of 0.7°C/bit. Once temperature changes, the sensor measures the new temperature and updates the sensor register, which is connected to a range select logic circuit. TAR requires 1°C/bit temperature sensors, and the sensor register uses 6 bits to present $0 - 105$°C temperature range. Table 1.2 shows the truth table of the range select logic.

The implementation is depicted in Fig. 1.8, where the input shown in the second column of Table 1.2 is connected to the sensor register. The output of the range select logic is a 2-bit signal (the third column of Table 1.2) connected to the mode register. The mode register has N_{banks} sections, and the index of each section represents the ID of the bank. When a signal from the DRAM control unit enables the mode register, the corresponding section will

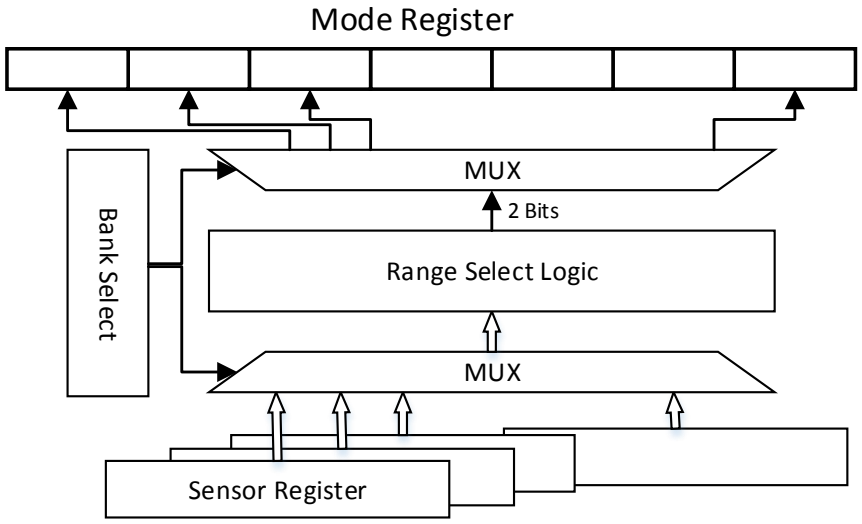
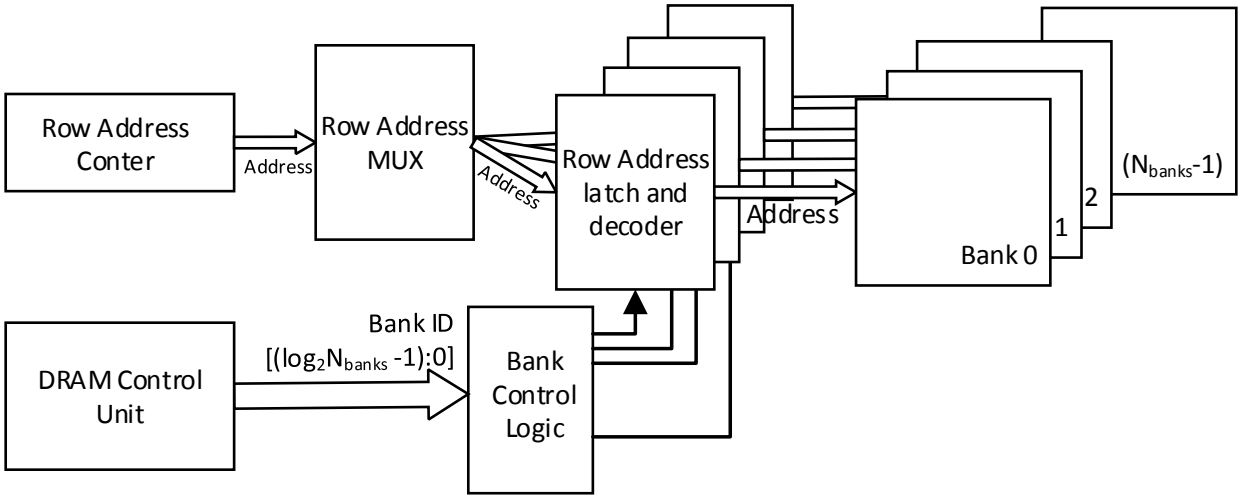


Figure 1.8: Range select logic converts sensor register bits into 2 bits mode register settings.

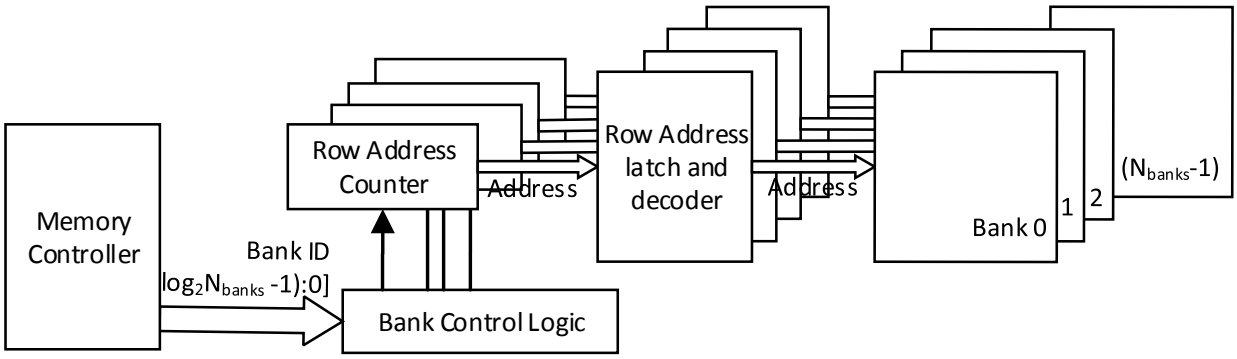
be updated by the range select logic. Since every bank has a unique section in the mode register, these bits represent not only the temperature range but also the ID of the bank.

1.4.2.2 Refresh Bank Control Logic and Row Selection Circuit

In conventional DRAM chips shown in Fig. 1.9(a), every rank has only one refresh counter which is connected to a row address MUX. The refresh counter sends its current address to the row address MUX. The address of the same group or rows received from the row address counter will be sent into the row address latch and decoder by the row address MUX. At the end, the bank control logic sequentially enables the row address latch and decoder which will then refresh the selected bank. In the proposed TAR shown in Fig. 1.9(b), the DRAM device can select the refresh group for each refresh bank, and the refresh bank selection function has been moved into the memory controller. When receiving the refresh signal from the refresh queue, the DRAM control unit blocks other requests for the refresh bank. Conventional DRAM refreshes the same group or rows for each bank, while the proposed TAR refreshes different groups or rows for different banks and only refreshes one bank for each refresh operation. This is because different banks have different refresh rates, so the



(a) Conventional Refresh Control Logic



(b) Temperature Aware Refresh Control Logic

Figure 1.9: Refresh bank control logic and row selection circuit.

rows that are recently refreshed are different. For example, the 95°C bank would require doubling the refresh rate as compared with the 85°C bank. In order to continuously refresh every group of rows within the same bank, the TAR utilizes N_{banks} row address counters to keep track of the last refreshed group of rows. All row address counters directly send the group or row addresses into the corresponding row address latch and decoder. Thus, there is no need to use row address MUX. In addition, the bank control logic decodes the bank ID and only enables the selected row address counter. The group or row addresses will then be sent into the row address latch and decoder to refresh the corresponding bank. After finishing the refresh operation for this bank, the row address counter increases to the next group address.

1.5 Results and Discussion

1.5.1 Simulation Platform and System Configuration

To evaluate the proposed TAR technique, we use Gem5 [22] as the system simulator and DRAMSim2 [23] as the memory controller and DRAM model. Gem5 provides a cycle-accurate processor model which builds up the whole simulated system for running various benchmarks. It is well connected with the cycle-accurate DRAMSim2 memory model, which implements the proposed TAR technique. Table 1.3 shows the system configurations.

The main parameter of refresh operation for DRAM is tRFC. The tRFC of all-bank refresh policy for 8G, 16G, and 32G DRAM is 350ns, 480ns, and 640ns, respectively. In this policy, the tREFI signal refreshes the entire banks in a memory rank, so that this rank cannot process any other memory requests. This enables all banks to refresh at a pipelined order. On the other hand, in the per-bank refresh policy, which is mainly used in mobile devices, a DRAM chip refreshes a single bank while the other banks in the same rank are available to process memory requests; but the banks in the same rank cannot be refreshed

Table 1.3: System Configurations

Number of cores	4
Processor frequency	1GHz
Cache line size	64B
Last level cache	1MB
Associativity for Last level cache	8-way
DRAM Type	DDR3
Memory clock frequency	667MHz
Number of channels	4
Number of ranks	2/8G, 1/16G, 1/32G per channel
Number of banks	8 per rank
Row Buffer Policy	open page
tRFCab	350ns/8G, 480ns/16G, 640ns/32G
tRFCpb	152ns/8G, 208ns/16G, 287ns/32G

at the pipelined order. Thus, in the pre-bank policy, the tRFC does not equal to all-bank's tRFC/N_{banks}. For example, in a 2G LPDDR2 device the all-bank tRFC is 130ns and the pre-bank tRFC is 60ns [25], which is 2.3× smaller. This ratio has been used by many existing work [17] to project the tRFC for DRAM with a larger size, which is also adopted in this chapter for 16G and 32G DRAM devices. We also use this ratio to derive the pre-bank tRFC of 152ns, 208ns, and 278ns for 8G, 16G, and 32G DRAM, respectively. We calculate the refresh interval for 45°C and 105°C based on the retention time given by [18]. As shown in Table 1.1, we also use 45°C, 85°C, 95°C, 105°C as the interval settings for four temperature ranges: $0-45$°C, $45-85$°C, $85-95$°C, and $95-105$°C. Temperature higher than 105°C is not considered as it occurs rarely.

We use the benchmarks from SPEC CPU2006 [26] to run simulations and evaluate the performance in terms of instruction per cycle (IPC). Our 3D-stacked DRAM structure assumes that different DARM layers are stacked on the top of the logic layer. Every layer has one channel which has one rank for 16G and 32G DRAM and two ranks for 8G DRAM. We adopt the DRAM thermal distribution as derived in [2], [6] for different benchmarks.

1.5.2 TAR Technique Compared with All-Bank Refresh

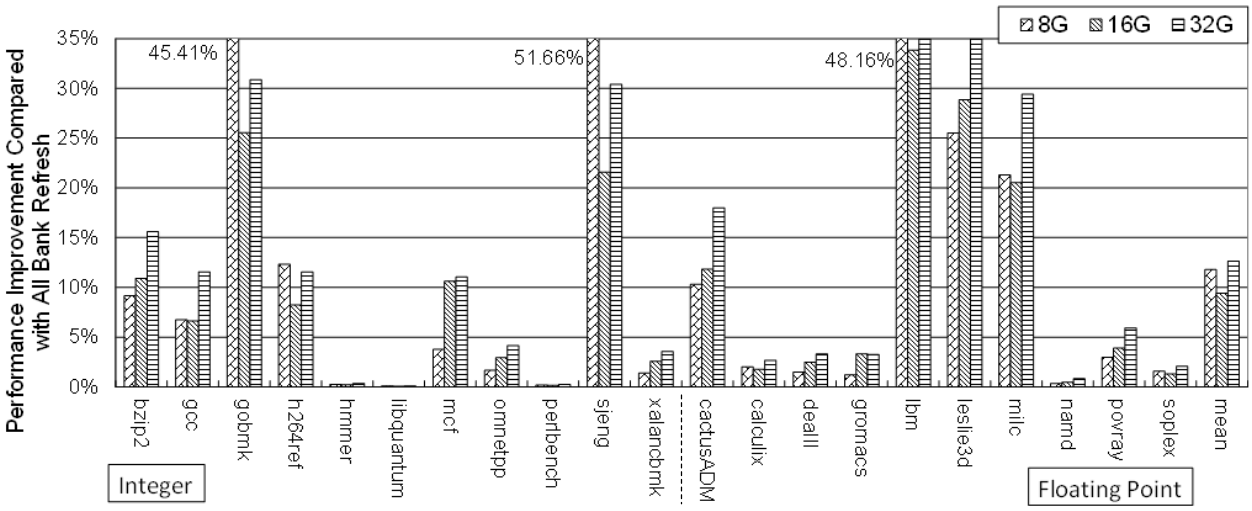


Figure 1.10: TAR performance improvement compared with All-bank refresh

Fig. 1.10 shows the performance improvement between TAR and the all-bank refresh policy for different DRAM capacities. The average improvement for 8G, 16G, and 32G DRAM is 11.78%, 9.41%, and 12.63%, respectively. The reason that the 8G DRAM has a larger improvement than the 16G DRAM is that *gobmk, lbm, sjeng*, and *h264ref* benchmarks have large gain differences between 8G and 16G DRAM configurations. The 8G DRAM has 2 ranks in each channel, which means there are 2 ranks in the same layer. Thus, using TAR, there is only one bank out of 16 banks working at the peak refresh rate. In conventional all-bank refresh, all banks and ranks are unavailable for memory requests during the refresh operations, which are all running at the high refresh rate. In contrast, except for the refreshing banks, other banks in the TAR are accessible for read/write operations and only one bank has the high refresh rate. As a result, the proposed TAR significantly improves the performance and the performance improvement in general increases as the DRAM capacity gets larger.

1.5.3 TAR Technique Compared with Per-Bank Refresh

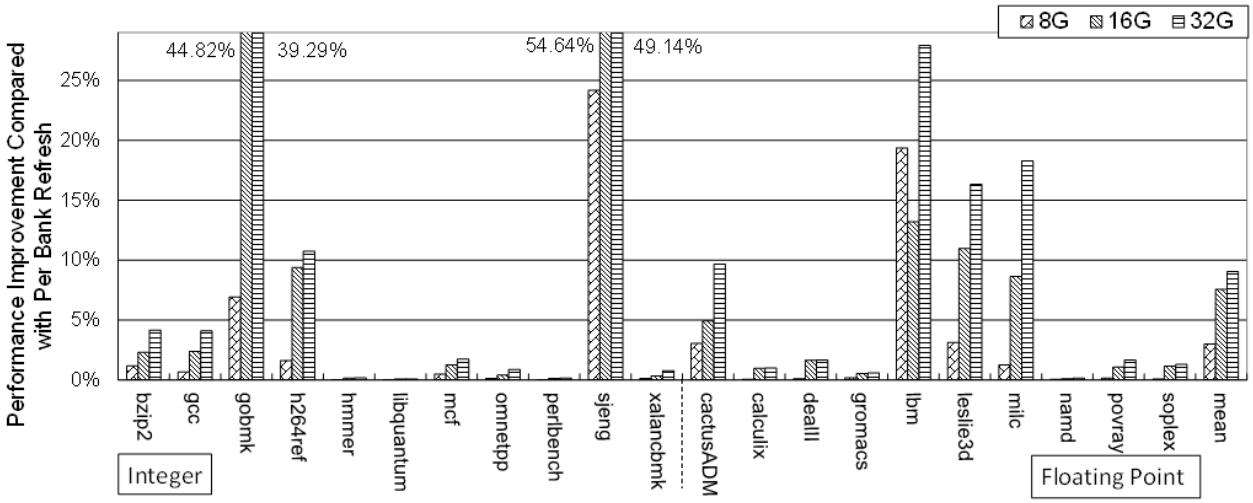


Figure 1.11: TAR performance improvement compared with Per-bank refresh

The performance improvement of TAR compared with the per-bank refresh policy is shown in Fig. 1.11. The improvement is a little bit smaller than that compared with the all-bank refresh policy, i.e., the average improvement for 8G, 16G, and 32G DRAM is 2.98%, 7.57%, and 9.03%, respectively. This is because per-bank refresh has already enabled the access for banks that are not refreshing. Thus, the margin for performance improvement is reduced. Nevertheless, the performance gain of the proposed TAR is still favorable in 3D IC designs.

1.6 Conclusion

The hot spots in 3D structures introduce a major performance penalty for 3D-stacked DRAM, as the refresh rate of all DRAM banks is determined by the peak temperature. As the capacity of DRAM increases further, the refresh operation will consume more time, which blocks memory read and write accesses and affects the system performance. In this

chapter, we have introduced the *Temperature Aware Refresh* (TAR) technique to mitigate this problem for 3D-stacked DRAM. The proposed TAR makes each bank refresh at different rates depending on the actual thermal conditions they are experiencing. In addition, each bank is able to adjust to the new refresh rate when the temperature changes. Simulation results show that the proposed TAR not only improves the performance as compared to the all-bank refresh policy but also has a better performance compared with the per-bank refresh policy for mobile platforms.

Chapter 2

Exploiting Early Tag Access for Reducing L1 Data Cache Energy in Embedded Processors

2.1 Introduction

DRAM based memory require refresh operation to maintain data, but SRAM based memory is able to keep the data without any refreshment. However, power consumption is still a problem for SRAM based memories such as cache memory. Reducing power consumption in cache memory is a critical problem as well for embedded processors that target low-power applications. It was reported that on-chip caches could consume as much as 40% of the total chip power [27], [28]. Furthermore, large power dissipation could cause other issues such as thermal effects and reliability degradation. The thermal degradation will cause negative impact on memory system performance. This problem is compounded by the fact that data caches are usually performance critical. Therefore, it is of great importance to reduce cache energy consumption while minimizing the impact on processor performance.

Many cache design techniques [29]–[38] have been proposed at different levels of the design abstract to exploit the tradeoffs between energy and performance. As caches are typically set-associative, most microarchitectural techniques aim at reducing the number of tag and data arrays activated during an access, so that cache power dissipation can be reduced. Phased caches [39] access tag arrays and data arrays in two different phases. Energy consumption can be reduced greatly because at most only one data array corresponding to the matched tag, if any, is accessed. Since the phased caches require additional access cycles, they are usually applied in the lower level memory such as L2 caches, whose performance is relatively less critical. For L2 caches under the write-through policy, a way-tagging technique [40] sends the L2 tag information to the L1 cache when the data is loaded from the L2 cache. During the subsequent accesses, the L2 cache can be operated in an equivalent direct-mapping manner,

thereby improving the energy efficiency without incurring performance degradation. To reduce the energy consumption of L1 caches, way-predicting techniques [41]–[48] make a prediction on the tag and data arrays that the desired data might be located. On the condition that the prediction is correct, only one way will be accessed to complete the operation; otherwise all ways are accessed to search for the desired data. Mis-predictions lead to cache re-accesses, which introduce a performance penalty. For very high set-associative (e.g., 32-way) data caches in high-end microprocessors, a technique proposed in [34] employs an additional fully associative memory to record the way information of the recent cache accesses at the Load/Store Queue (LSQ) stage. This kind of memory will be searched before a cache access, and the potential destination way can be determined if the address is found. This technique, however, may not be effective for typical L1 data caches (e.g. 4-way set-associative) commonly used in embedded processors due to the overhead associated with the additional fully associative memory. Content-addressable-memory (CAM) is considered as an option for low power cache design as well [36], [37]. However, its access latency is longer than the conventional SRAM-based tag array when the assocaitivity of a cache is low, e.g. 4 or 8, which is the typical for most high performance embedded systems.

In this chapter, we propose a new cache technique, referred to as *Early Tag Access* (ETA) cache, to improve the energy efficiency of L1 data caches. In a physical tag and virtual index cache, a part of the physical address is stored in the tag arrays while the conversion between the virtual address and physical address is performed by the Translation-Lookaside-Buffer (TLB). By accessing tag arrays and TLB during the LSQ stage, the destination ways of most memory instructions can be determined prior to accessing the L1 data cache. As a result, only one way in the L1 data cache needs to be accessed for these instructions, thereby the energy consumption will be significantly reduced. Note that the physical addresses generated from the TLB at the LSQ stage can also be used for subsequent cache accesses. Therefore, for most memory instructions, the energy overhead of way determination at the LSQ stage can be compensated for by skipping the TLB accesses during the cache access stage. For

memory instructions whose destination ways cannot be determined at the LSQ stage, an enhanced mode of ETA is proposed to reduce the number of ways accessed at the cache access stage. Note that in many high-end processors, accessing L2 tags is done in parallel with the accesses to the L1 cache [53]. Our technique is fundamentally different as early tag assesses are performed at the L1 cache.

While many high-end processors utilize parallel LSQ and L1 data cache accesses for performance improvement at the cost of high cache traffic and energy consumption, the proposed ETA cache is more effective for embedded processors, where the accesses to the LSQ and L1 data cache are typically performed in series [50], [51] to take advantage of load forwarding for reducing cache traffic and energy consumption. Note that the proposed ETA cache may also be applied to some general purpose processors. For example, some processors (e.g., Alpha 21264 [49]) have a dispatch stage during the Load/Store phase, at which the effective addresses of memory operations are available. Thus, upon accessing the L1 data cache, the available destination way can be utilized to reduce the cache energy consumption. Compared with existing techniques, the proposed ETA cache enables better tradeoffs between energy efficiency and performance.

2.2 Background

To reduce conflict misses, set-associative architecture is commonly employed in cache design. Fig. 2.1 shows a simple two-way set-associative cache and TLB, where the tag and data arrays are the two major components. In the conventional L1 data cache, all tag arrays and data arrays are activated simultaneously for every read/write access to reduce the access latency. Typically, the latency of the L1 data cache is one clock cycle, though this latency could be higher in deeply pipelined processors. On the other hand, accesses to the tag arrays can always be finished in one cycle [53], [54]. Due to the temporal/spatial locality inherent in various programs [57], data will stay for a while once they have been brought into the

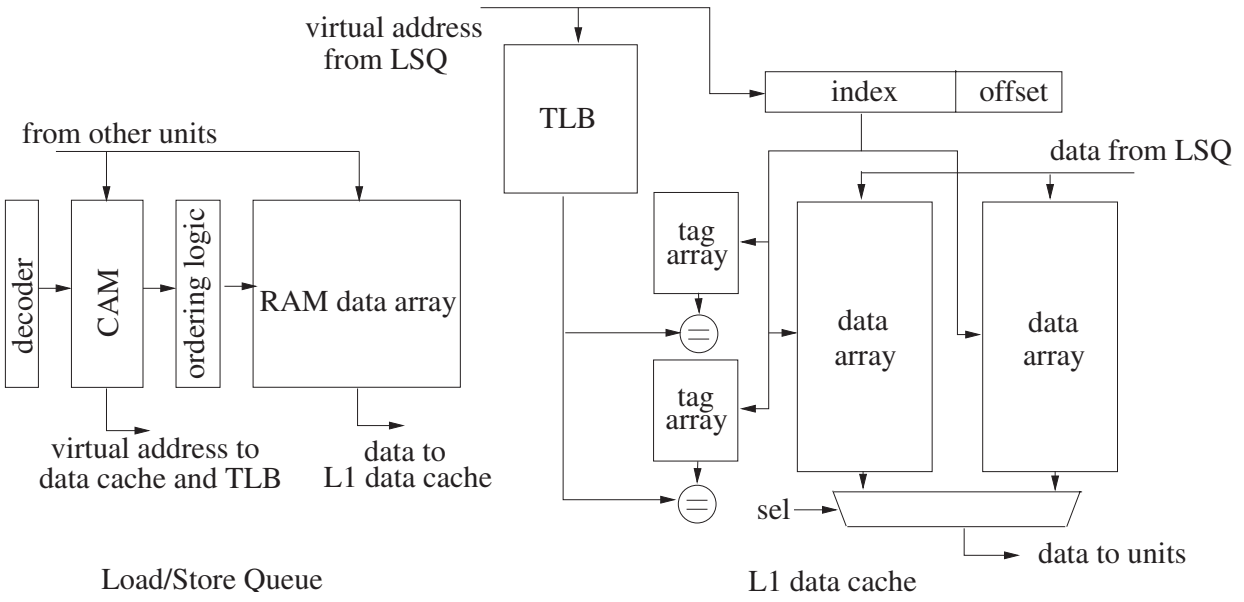


Figure 2.1: Conventional architecture of LSQ and L1 data cache.

data cache. This implies that the tag arrays will keep their contents unchanged except when a cache miss occurs.

Fig. 2.2 shows a portion of the pipeline between the address generation stage and the memory stage in a typical embedded processor, where the LSQ and L1 data cache are accessed in series [50], [51] to take advantage of load forwarding for reducing cache traffic and energy consumption. A load/store instruction will be sent to the LSQ before being issued to the data cache. Meanwhile, the instruction is compared with the existing ones in the LSQ to determine whether the instruction can be issued to the data cache at the next clock cycle. If not, the instruction will stay in the LSQ stage for more than one clock cycle.

2.3 Early Tag Access (ETA)

2.3.1 Motivation

From the existing cache architecture, we observe that (1) the memory address of a load/store will be available in the LSQ stage for at least one clock cycle before being is-

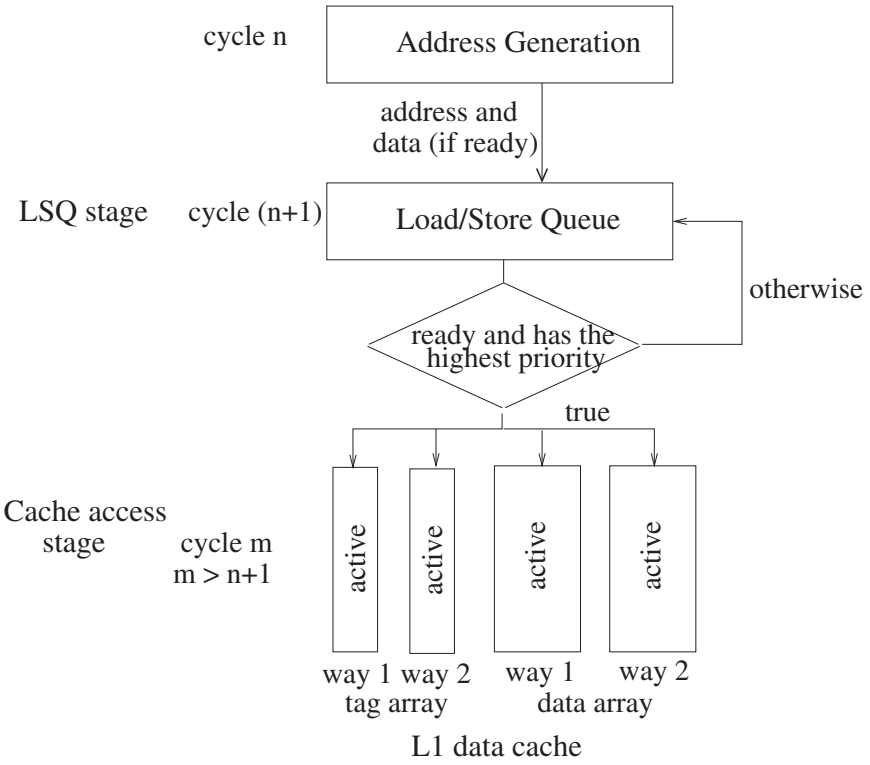


Figure 2.2: Pipeline of a load/store instruction between LSQ and L1 data cache.

sued to the data cache, while the access to the tag arrays can be finished in one cycle; and (2) due to the temporal/spatial locality, the tag arrays will not be updated until a cache miss occurs. Since in the conventional architecture the destination way of a memory instruction cannot be determined before the cache access stage, all the ways in the L1 data cache need to be activated during a cache access for performance consideration at the cost of energy consumption. Based on these observations, we propose a new cache technique to improve the energy efficiency of L1 data caches.

2.3.2 The Proposed ETA Cache

In a conventional set-associative cache, all ways in the tag and data arrays are accessed simultaneously. The requested data, however, only resides in one way under a cache hit. The extra way accesses incur unnecessary energy consumption. In this section, a new cache

architecture referred to as *Early Tag Access* (ETA) cache will be developed. The ETA cache reduces the number of unnecessary way accesses, thereby reducing cache energy consumption. To accommodate different energy and performance requirements in embedded processors, the ETA cache can be operated under two different modes: the basic mode and the advanced mode.

2.3.2.1 The Basic Mode

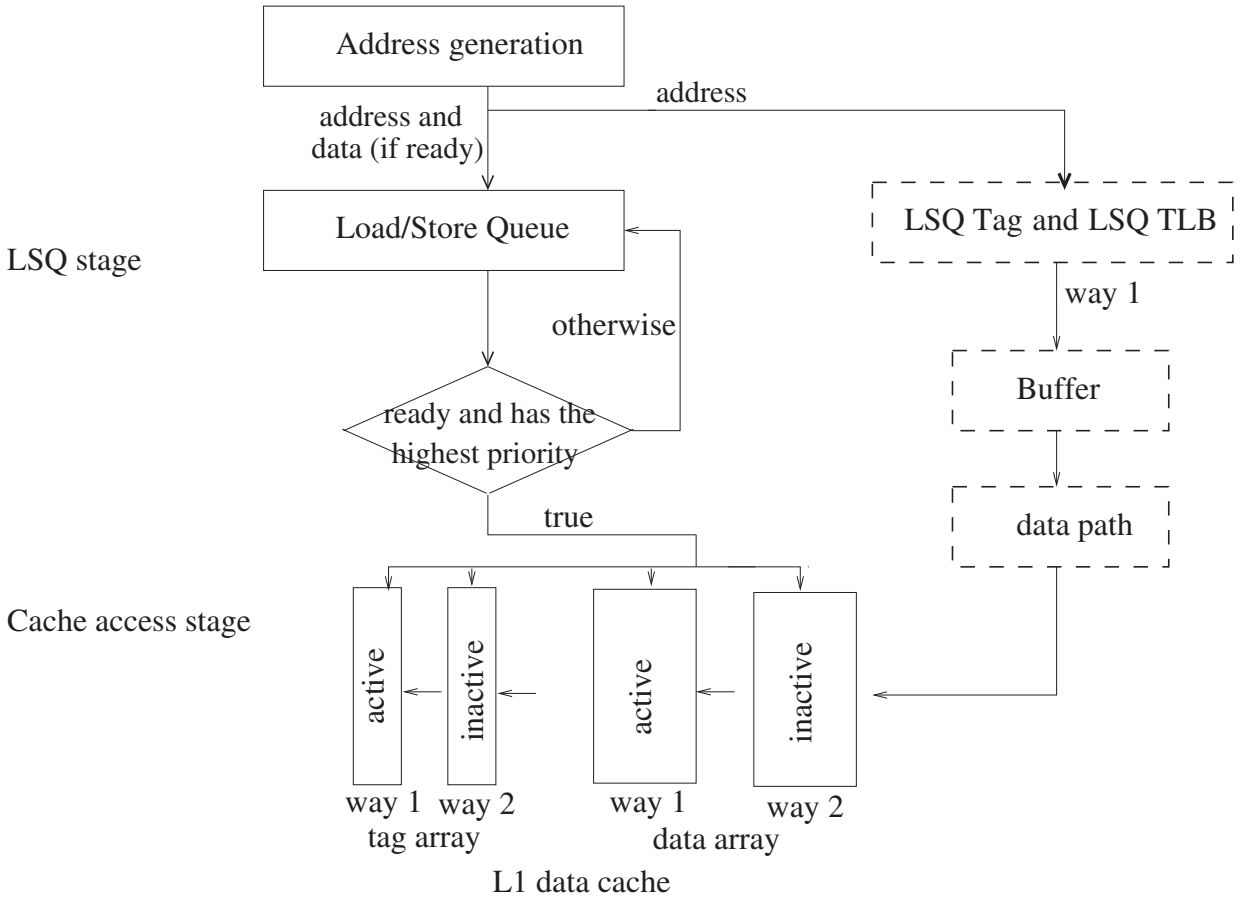


Figure 2.3: The operation of a load/store instruction between LSQ and L1 data cache under the proposed ETA cache.

It is possible to perform an access to the tag arrays at the LSQ stage due to the availability of memory addresses. In the basic mode of the ETA cache, each time a memory instruction

is sent into the LSQ, an access to a new set of tag arrays and TLB (referred to as LSQ tag arrays and LSQ TLB, see Section 2.4.1) is performed, as shown in the dotted lines in Fig. 2.3. This new set of LSQ tag arrays and LSQ TLB are implemented as a copy of the tag arrays and TLB of the L1 data cache, respectively, to avoid the data contention with the L1 data cache. If there is a hit during the LSQ lookup operation, the matched way in the LSQ tag arrays will be used as the destination way of this instruction when it accesses the L1 data cache subsequently. If this destination way is correct, only one way in the L1 data cache needs to be activated and thus enables energy savings. On the other hand, if a miss occurs during the lookup operation in the LSQ tag arrays or in the LSQ TLB, the L1 data cache will be accessed in a conventional manner, i.e., all ways in the tag arrays and data arrays of the L1 data cache will be activated. From Fig. 2.3, we can see that the two sets of tag arrays and TLB are accessed at two different stages: LSQ stage and cache access stage. To differentiate these accesses, we use the terminology defined in Table 2.1.

Table 2.1: Definitions of cache access in this chapter.

	LSQ stage	cache access stage
tag array access	early tag access	actual tag access
TLB access	early TLB access	actual TLB access
tag hit/miss	early tag hit/miss	cache hit/miss
TLB hit/miss	early TLB hit/miss	TLB hit/miss
destination way	early destination way	actual destination way

It is possible that the early destination way of a memory instruction determined at the LSQ stage is not the same as the actual one determined at the cache access stage due to the cache misses that happen in between. This causes a cache coherence problem if the memory instruction simply follows its early destination way, if any, to access the L1 data cache. To illustrate this, consider a simple example shown in Fig. 2.4. Assume that an instruction $Inst1$ loads $data1$ into register $R1$. At the LSQ stage, $data1$ is stored in the $way0$ of the

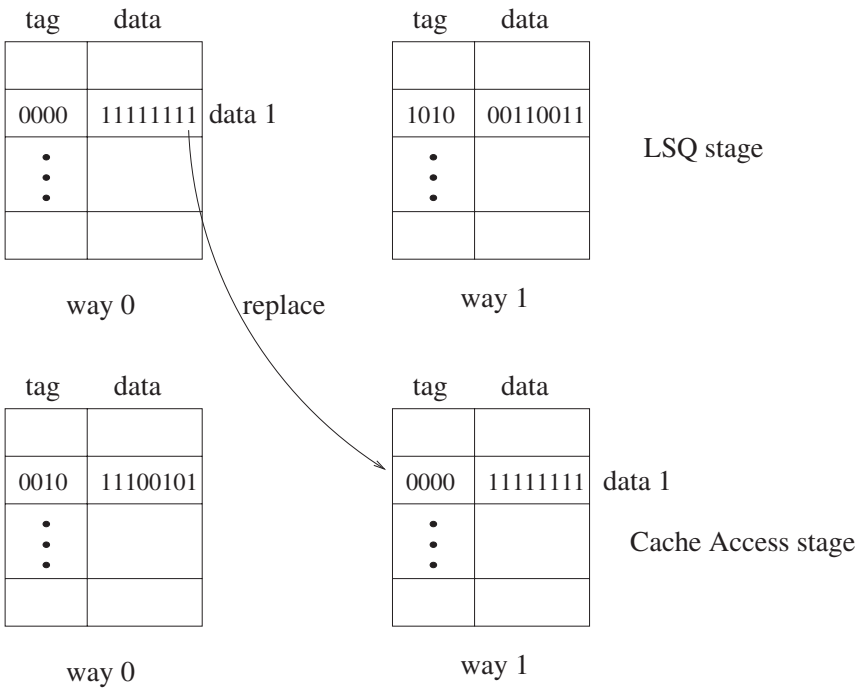


Figure 2.4: Illustration of cache coherence problem related to early destination way information.

L1 data cache. Thus $Inst1$ has an early hit at the LSQ stage and its early destination way is determined as $way0$. When $Inst1$ accesses the L1 data cache at the cache access stage (which may happen several clock cycles later if some instructions in the LSQ have a higher priority than $Inst1$), $data1$ actually resides in the $way1$ of the L1 data cache due to a cache miss that happens between the LSQ stage and cache access stage of the instruction $Inst1$. In this case, a cache coherence problem will occur if $Inst1$ simply uses its early destination way to access the L1 data cache. To avoid this problem, the ETA cache in the basic mode requires the memory instruction to access all the ways in the actual tag arrays during the cache access stage. During the same time, the data arrays are also accessed in parallel using the early destination way only (as the actual destination way is not available yet) to reduce energy consumption while maintaining performance. The destination way obtained from the actual tag arrays is then compared with the early destination way to detect any cache coherence problem. If a cache coherence problem is detected, the accessed data from

the data arrays is discarded and an additional access is performed. This occurs rarely, e.g, less than 1% of the total cache accesses. Therefore, only minor performance degradation is incurred, which is typically acceptable for embedded processors.

Note that another method to resolve the above cache coherence problem is to only access the predicted early destination way during the cache access stage. If the prediction is incorrect, simultaneous tag-data lookup will be performed to obtain the desired data from the data arrays. This solution saves energy in accessing tag arrays but consumes more energy during the data array access when the prediction is incorrect. Since data arrays consume much larger energy than tag arrays (see Table 2.2), we choose to use the first method in this chapter. However, if the re-access rate is extremely small, the second method may enable 1% additional energy savings due to the reduction in accessing the tag arrays.

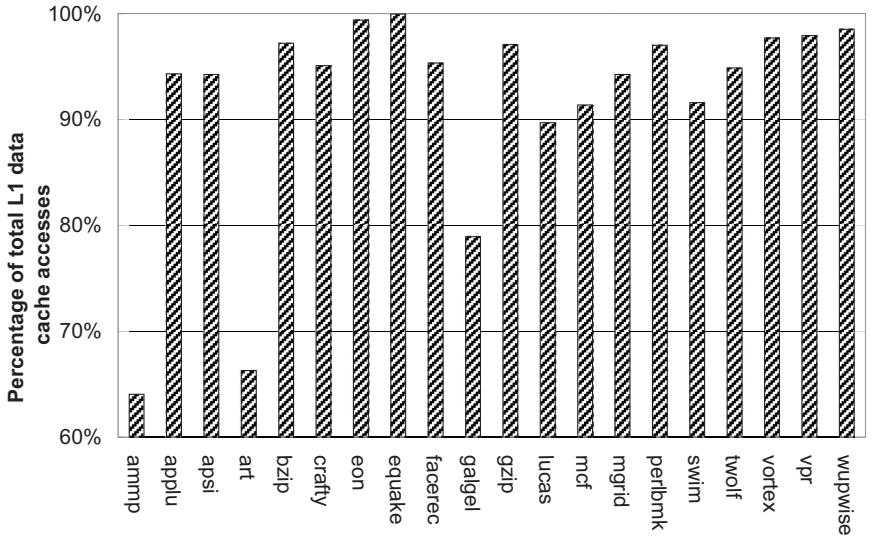


Figure 2.5: Percentage of cache accesses whose destination ways can be determined through early tag accesses at the LSQ stage under SPEC2000 benchmarks.

To understand the effectiveness of the ETA cache under the basic mode, we evaluate the SPEC CPU2000 benchmarks [55] as well as the embedded benchmark suite MiBench [56]. Simulation results from the *Simplescalar* toolset [52](see Section 2.5 for the system configuration) show that even in the worst case of the SPEC CPU2000 benchmarks (e.g., ammp),

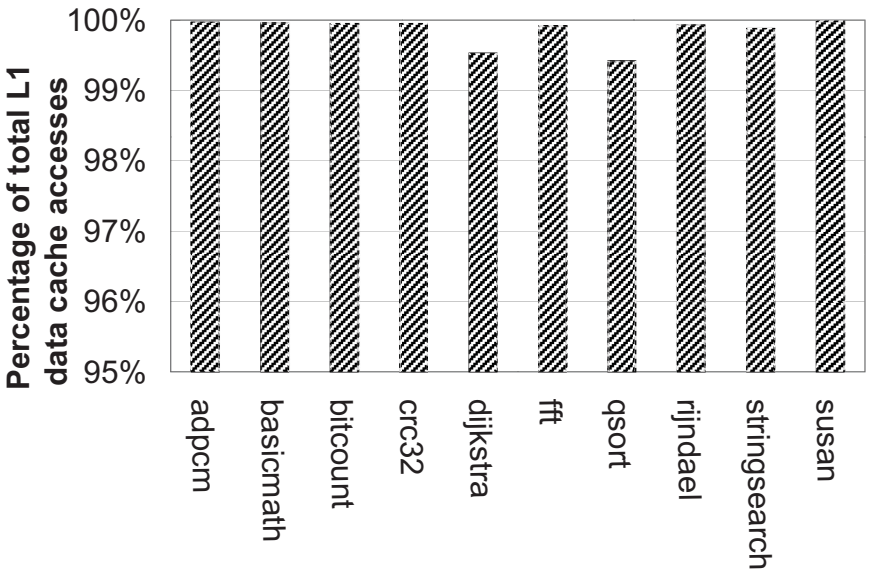


Figure 2.6: Percentage of cache accesses whose destination ways can be determined through early tag accesses at the LSQ stage under MiBench benchmarks.

more than 60% of the actual destination ways match early destination ways (see Fig. 2.5). Fig. 2.6 shows even better results for the MiBench benchmarks, due to their high hit rates (more than 99.5%) in the L1 data cache. Thus, most memory instructions can utilize their early destination ways determined at the LSQ stage to access the L1 data cache.

2.3.2.2 The Advanced Mode

From Figs. 2.5 and 2.6, there are some memory instructions whose early destination ways cannot be determined due to either early tag misses or early TLB misses. This could be significant for applications with a high miss rate (e.g., ammp and art). As shown in Figs. 2.7 and 2.8, these early tag/TLB misses at the LSQ stage are usually associated with real misses at the cache access stage. In fact, from Figs. 2.5–2.8, we observe that only a few memory instructions have early tag/TLB misses but real cache hits.

This observation leads to the development of the advance mode of the ETA cache for early tag/TLB misses (i.e., early destination ways cannot be determined at the LSQ stage). Specifically, when a memory instruction with either an early tag miss or an early TLB miss

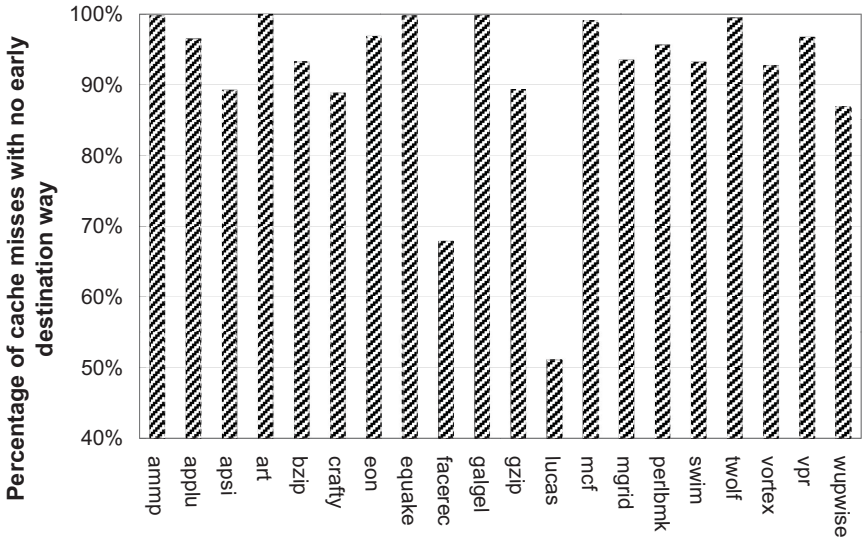


Figure 2.7: Percentage of cache misses with no early destination ways available under SPEC2000 benchmarks.

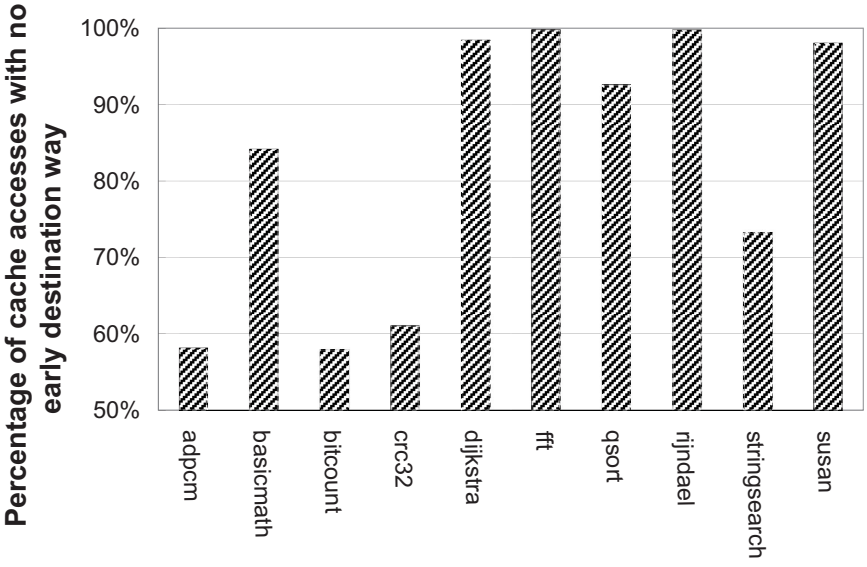


Figure 2.8: Percentage of cache misses with no early destination ways available under MiBench benchmarks.

accesses the L1 data cache, the tag arrays of the L1 data cache are accessed first. In most cases this will be a real cache miss, and thus the L1 cache access is completed without the need to access the data arrays of the L1 data cache, thereby saving energy. In rare cases if a cache hit occurs, the actual destination way is obtained from the tag arrays and only this way is accessed in the data arrays at the next clock cycle. By applying this advanced mode, the energy consumption can be reduced because most of the memory instructions with early tag/TLB misses do not need to access the data arrays.

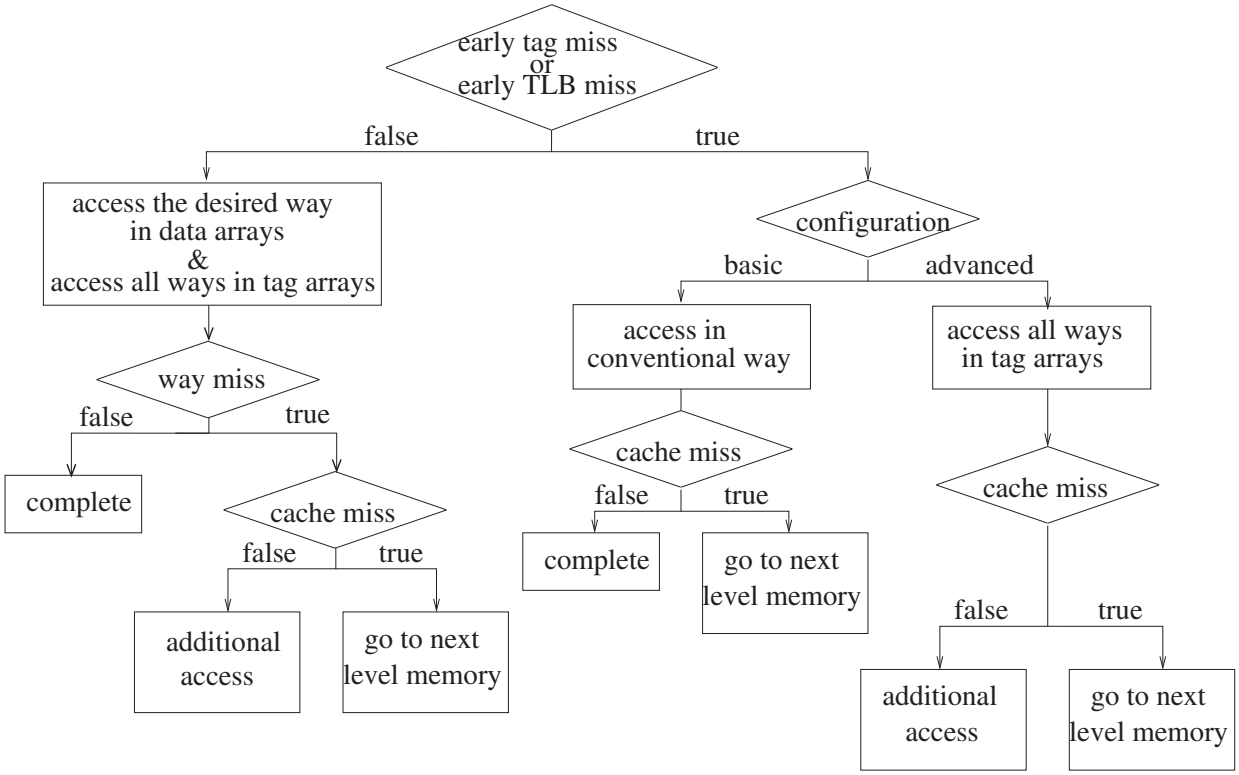


Figure 2.9: Operation flow of the proposed ETA cache under two modes.

Different from the phased cache [39] where all data cache accesses are divided into two phases, the ETA cache only applies the advanced mode to memory instructions with early tag and/or early TLB misses. Some of these instructions have real cache hits, thereby causing an extra cycle in the cache access. Since the number of such cases (i.e., early tag/TLB misses but

real cache hits) is very small, the performance overhead is negligible. Note that the advanced mode can be turned off if performance is truly critical. The two-mode configuration of the proposed ETA cache is summarized in Fig. 2.9.

2.4 Implementation

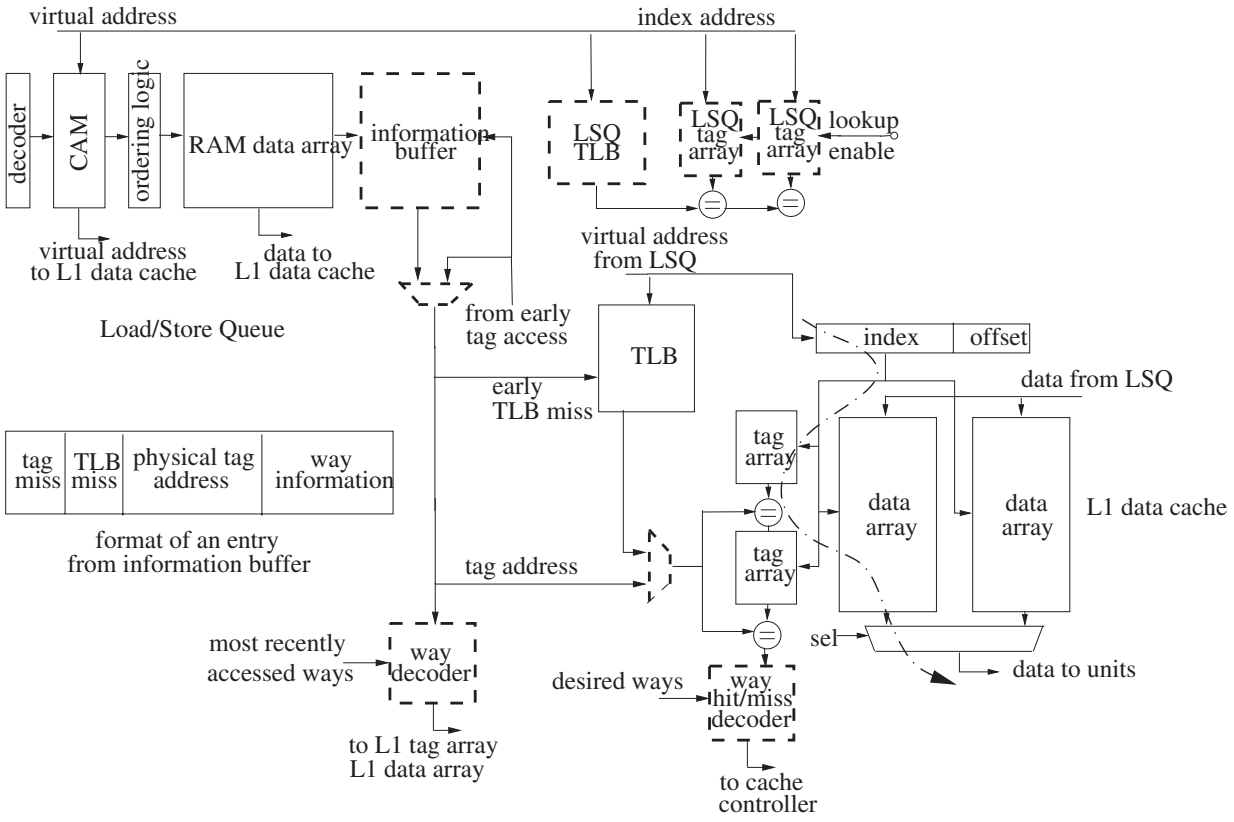


Figure 2.10: The proposed ETA cache (blocks in dot line are new components).

This section presents the VLSI implementation of the proposed ETA cache. Fig. 2.10 depicts a two-way set-associative L1 data cache for demonstration. The key components in the ETA cache, such as LSQ tag arrays, LSQ TLB, information buffer, way decoder, and way hit/miss decoder will be discussed in the following sections.

2.4.1 LSQ Tag Arrays and LSQ TLB

To avoid the data contention with the L1 data cache, the LSQ tag arrays and LSQ TLB are implemented as a copy of the tag arrays and TLB of the L1 data cache, respectively. There are two types of operations in the LSQ tag arrays and LSQ TLB: lookup and update. Each time a memory address reaches the LSQ, the LSQ tag arrays and LSQ TLB will be searched for the early destination way. In case of a hit, the early destination way will be available; otherwise, the instruction will cause either an early tag miss (if the access to the LSQ tag arrays encounters a miss) or an early TLB miss (if the address is not in the LSQ TLB). For update operations, the contents of LSQ tag arrays and LSQ TLB are updated with the tag arrays and TLB of the L1 cache, so that they are identical to avoid cache coherence problems. The update logic of LSQ tag arrays and LSQ TLB is the same as that of the tag arrays and TLB of the L1 cache.

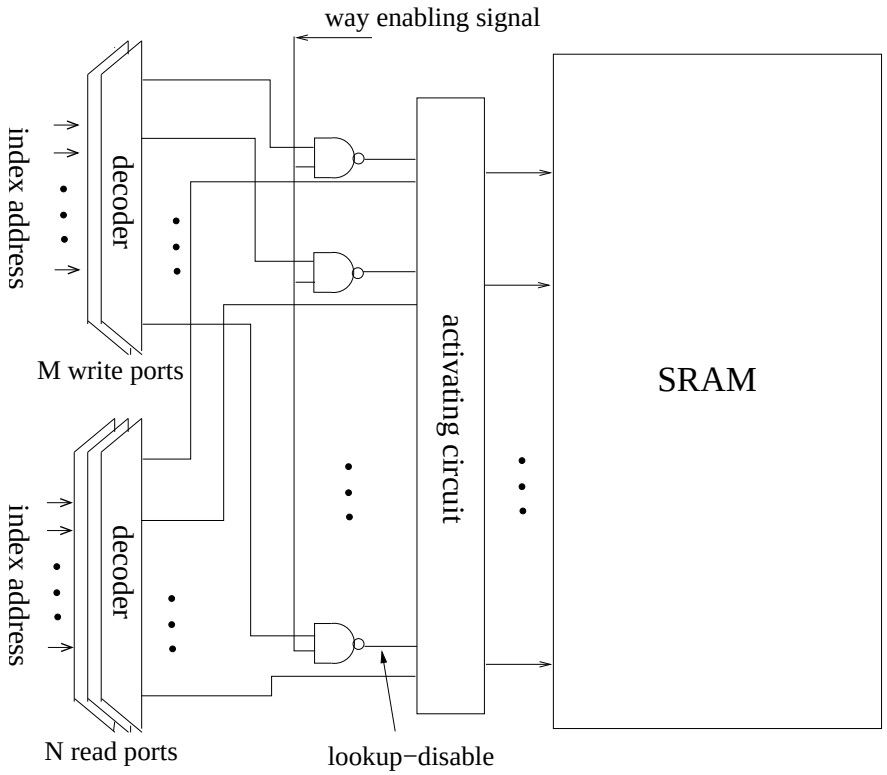


Figure 2.11: Implementation of LSQ tag arrays (only one way is shown for simplicity).

Fig. 2.11 shows the implementation of the LSQ tag arrays, where only one way is shown as the other ways are the same. Consider that generally at most N instructions can enter the LSQ while the L1 data cache allows M replacements to occur at the same time. Therefore, there might be at most N lookup operations and M update operations occurring at the LSQ tag arrays and LSQ TLB at the same time. In order to perform these operations simultaneously, the LSQ tag arrays and LSQ TLB have N read ports and M write ports. In the simulations in Section 2.5, both M and N are chosen to be two for the purpose of demonstration. Write/read conflicts occur when the lookup and update operations target the same location of the LSQ tag arrays at the same time. To address this issue, we disable the lookup operation if an update operation is currently performed. This is achieved by the control signal lookup-disable, which is generated by the way enabling signals from the cache controller for cache replacements. Consider a two-way set-associative cache for example. Assume that there is a replacement occurring at the way 1 of the L1 data cache. As a result, the way enabling signal is set to "1" and then sent to the NAND gates in the way 1 of the LSQ tag arrays. If the write decoder outputs a "0," i.e., no update operation on this entry of the tag array, the lookup-disable signal will be set to "1" and the activating circuit will not block the lookup operation on this entry. Otherwise the lookup-disable signal will be "0," and the activating circuit will block possible lookup operations to avoid write/read conflicts. The lookup operation, if any in this case, is considered as a miss. This miss might cause some performance degradations if it turns out to be a cache hit in the cache access stage. Fortunately, this rarely happens. We observed from simulations that less than 0.01% of the total LSQ accesses experienced this issue. Note that if the way enabling signal is "0," i.e., no update operation, the lookup operation will not be affected. This scheme is also used in the LSQ TLB. Since the activating circuit introduces no performance penalty [28], the enabling circuit increases the critical path of the LSQ tag arrays and LSQ TLB by the delay of an NAND gate. This delay is trivial as compared with the critical path of the L1 data cache [58]. Thus, no new critical path is introduced by the proposed scheme.

Note that a larger issue width would need more read and write ports in the LSQ tag arrays and LSQ TLB, and thus the overhead will increase. However, the overhead relative to the size of L1 cache and TLB is more or less the same. This is because the number of the read and write ports in the tag arrays and TLB of the L1 cache will increase as well. Also, the size of the LSQ tag arrays and LSQ TLB is very small as compare to the data arrays.

2.4.2 Information Buffer

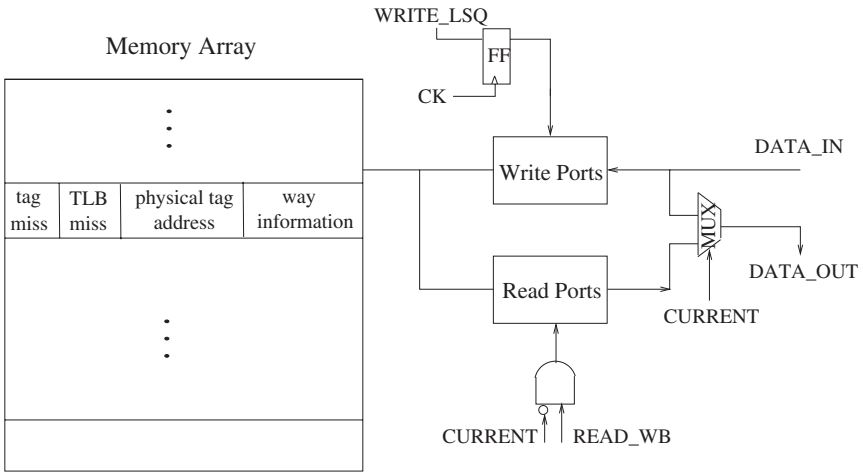


Figure 2.12: Implementation of information buffer.

As shown in Fig. 2.2, a load/store instruction might stay in the LSQ stage for more than one clock cycle. This requires an information buffer to hold the early destination way until the corresponding load/store instruction is issued to the L1 data cache. The implementation of the information buffer is shown in Fig. 2.12. It has the same number of entries as the LSQ. In each entry, the tag miss and TLB miss bits ("1" for hit and "0" for miss) indicate the status of early tag and early TLB accesses, respectively. The early destination way is stored in the way information bits. The physical tag address is generated by searching the LSQ TLB. For a memory instruction with an early tag hit, the access to the actual TLB during the cache access stage can be avoided by sending this physical tag address to the data

cache. Since the information buffer is small (32 entries with 20 bits per entry in a typical configuration), the induced energy overhead can be easily offset by the energy savings from the cache access stage.

The information buffer has separate write and read ports to support parallel write and read operations. The write operations of the information buffer always start one clock cycle later than the corresponding write operations in the LSQ. This is because the accesses to the LSQ, LSQ tag arrays, and LSQ TLB occur simultaneously. Since the way information is available after the write operations in the LSQ, this information will be written into the information buffer one clock cycle later than the corresponding write operation in the LSQ. Thus, the write signal of the information buffer can be generated by delaying the write signal of the LSQ by one clock cycle, as shown in Fig. 2.12.

When a load/store instruction is issued to the L1 data cache, both the instruction and its early destination way information are read out from the LSQ and information buffer, respectively, at the same time. However, simply using the same read signal for the LSQ and information buffer might lead to write/read conflicts in the information buffer. We use a signal *CURRENT* generated from the ordering logic in the LSQ as the enable signal for read operations. As shown in Fig. 2.12, when this signal indicates that an instruction just arrives at the LSQ, the read operation to the corresponding entry of the information buffer is disabled as the early destination way is not yet available.

2.4.3 Way Decoder and Way Hit/Miss Decoder

As shown in Fig. 2.9, during a cache access, all ways in the cache tag arrays and one way in the data arrays (i.e., the early destination way) are accessed if the instruction has an early tag hit. As explained in Section 2.3.2, activating all ways in the actual tag arrays is needed to detect the cache coherence problem. If a cache coherence problem is detected, an additional access to the L1 data cache is required. Here, we introduce a way hit/miss decoder

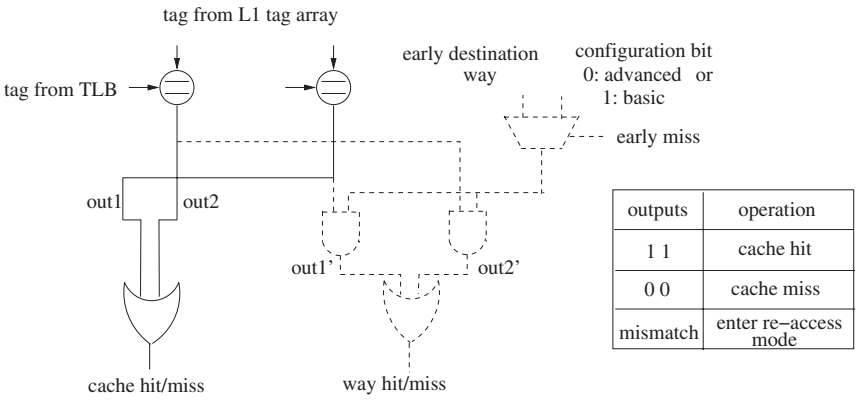


Figure 2.13: Implementation of way hit/miss decoder.

to determine whether the additional access is necessary. Fig. 2.13 shows the implementation of this decoder with dotted lines. A conventional cache hit/miss decoder is also shown with solid lines. The configuration bit is used to set the ETA cache for the basic mode or the advanced mode. As shown in Fig. 2.13, if both the cache hit/miss and way hit/miss signals indicate a hit (e.g. "1"), the cache access is considered a hit. The access will be treated as a miss if both the cache hit/miss and way hit/miss signals indicate a miss. If the cache hit/miss signal indicates a hit while the way hit/miss signal indicates a miss, or if the early destination way does not match the actual destination way, a cache coherence problem is detected. The cache controller will trigger an additional access in the data arrays at the next cycle based on the actual destination way. This access is referred to as re-access.

If the instruction incurs an early tag miss only, all ways in the tag arrays and data arrays of the L1 data cache need to be accessed if the cache works under the basic mode, while under the advanced mode only the tag arrays (all ways) need to be accessed. Note that under the advanced mode, the data arrays are accessed at the next clock cycle if the memory instruction leads to a real cache hit (see Section 2.3.2.2). In this case, a mismatch between the outputs of the cache and way hit/miss decoders occurs and subsequently triggers an additional access to the cache. If the instruction has an early TLB miss only, the access will be proceeded similarly except that the TLB needs to be accessed due to the lack of the

physical tag address.

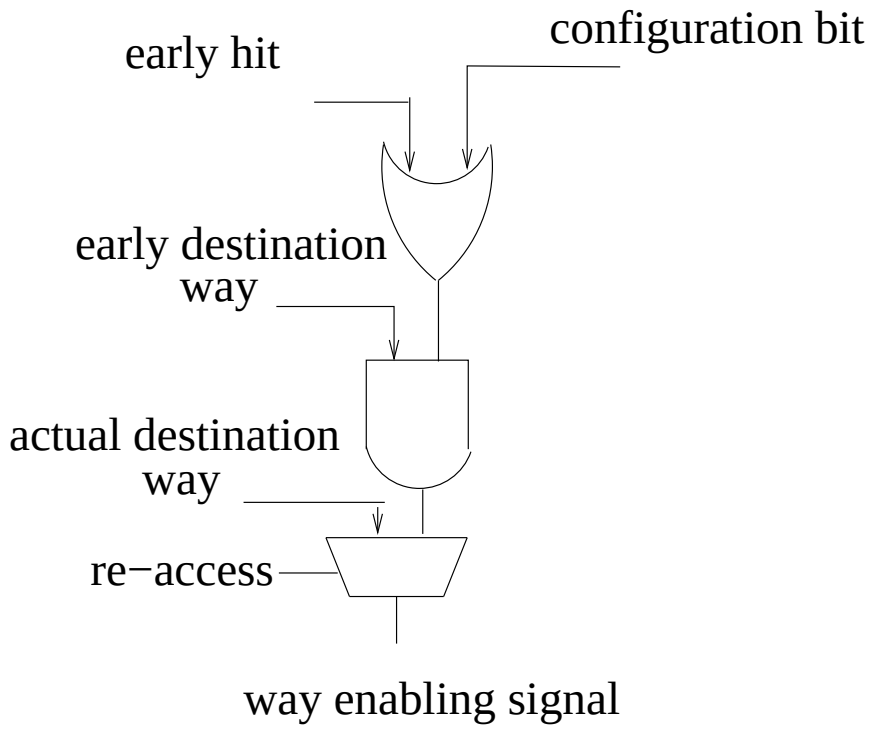


Figure 2.14: Implementation of way decoder.

In the proposed ETA cache, way enabling signals are needed to control the access to the ways in the data arrays. Fig. 2.14 shows the implementation of the way decoder that generates these signals. When the instruction is associated with an early hit (e.g., "1"), the data arrays need to be accessed according to the early destination way. If the instruction experiences an early tag miss or an early TLB miss, the configuration bit shown in Fig. 2.14 determines which way in the data arrays of the L1 data cache needs to be accessed. Specifically, by setting the configuration bit to "1," the ETA cache will operate under the basic mode. The outputs of the way decoder are all "1" because all the ways in the data arrays need to be accessed as discussed in Section 2.3.2.1. When the configuration bit is "0," i.e., advanced ETA mode, the way enabling signals are all "0" and the data array will be activated at the next cycle as discussed in Section 2.3.2.2. During a re-access, only the

actual destination way in the data arrays will be accessed. This is done by selecting the actual destination way from the MUX shown in Fig. 2.14. The actual destination way and the select signal "re-access" can be provided by the cache controller.

Overall, the proposed technique requires a new set of LSQ TLB and LSQ tag arrays, some changes in cache control logic, and additional hardware compared to the conventional cache. As discussed above, the control logic can be implemented by simple logic gates. Also, all the new components are not on the critical path as marked by the dotted line in Fig. 2.10. The hardware cost is very small. For example, considering a 16 K four-way set-associative L1 cache, the area overhead is around 2% of the L1 cache, which is similar to the way-halting cache proposed in [35]. This overhead is manageable at the physical design phase.

2.5 Evaluation

In this section, we evaluate the energy consumption, performance, and design overhead of the proposed ETA cache. We show experimental results for a processor with separate L1 instruction and data caches, both of which are dual ports 16 kB four-way set-associative with the cache line size of 64 bytes. The TLB is four-way set-associative with 32 sets and page size of 4 kB, and the LSQ is a unified LSQ with 32 entries. We also evaluate the energy efficiency of the proposed technique under different cache configurations, such as size and associativity. The Simplescalar toolset [52] is employed to obtain the performance measures. All the simulations are derived from the SPEC CPU2000 benchmarks [55] and MiBench benchmarks [56].

2.5.1 Energy Model

The following energy model is employed in our study:

$$
\begin{aligned}
\mathcal{E}_{tot} =& \mathcal{N}_{tag\_hit} \times (\mathcal{E}_{tag\_hit,L1} + \mathcal{E}_{info}) \\
&+ \mathcal{N}_{tag\_miss\_only} \times (\mathcal{E}_{tag\_miss\_only,L1} + \mathcal{E}_{info}) \\
&+ \mathcal{N}_{tag\_TLB\_miss} \times (\mathcal{E}_{tag\_miss,L1} + \mathcal{E}_{TLB} + \mathcal{E}_{info}) \\
&+ \mathcal{N}_{re-access} \times \mathcal{E}_{re-access,L1} \\
&+ \mathcal{N}_{tot} \times (\mathcal{E}_{info} + \mathcal{E}_{LSQ\_tag} + \mathcal{E}_{LSQ\_TLB}) \\
&+ \mathcal{E}_{others}, \qquad (2.1)
\end{aligned}
$$

where $\mathcal{N}_{tag\_hit}$, $\mathcal{N}_{tag\_miss\_only}$, and $\mathcal{N}_{tag\_TLB\_miss}$ are the numbers of load/store instructions with early tag hit, early tag miss only, and both early tag and TLB misses, respectively; and $\mathcal{N}_{re-access}$ is the number of cache re-accesses. The energy consumptions per access related to these cases are denoted as $\mathcal{E}_{tag\_hit,L1}$, $\mathcal{E}_{tag\_miss\_only,L1}$, $\mathcal{E}_{tag\_TLB\_miss,L1}$, and $\mathcal{E}_{re-access}$, respectively. $\mathcal{N}_{tot}$ is the total number of load/store instructions issued to the LSQ. $\mathcal{E}_{info}$, $\mathcal{E}_{LSQ\_tag}$, and $\mathcal{E}_{LSQ\_TLB}$ are the energy consumption per access of the information buffer, LSQ tag arrays, and LSQ TLB, respectively. Since the energy overheads from other components, such as the MUXes used in the ETA cache are very small, they are included in the $\mathcal{E}_{others}$. The energy overhead due to the update of LSQ tag arrays and LSQ TLB is also included in the $\mathcal{E}_{others}$ because update operations occur at a much lower rate than read operations (i.e., the miss rate is in general much lower than the hit rate). We employ CACTI 5.3 [59] to obtain the energy consumption per access under different operating modes using a 90-nm process. Since the proposed technique is technology-independent, these results are normalized by the energy consumption per access of the conventional L1 data cache for comparison. As shown in Table 2.2, the "access" columns summarize the number of ways that needs to be activated in different units. The energy consumptions of different components during different stages are also shown in Table 2.2. For example, the LSQ tag array and LSQ TLB are only accessed

during early tag access stage. Their total energy consumption per access is 21.4% of that of a conventional cache under the study. New components, such as way decoders, way hit/miss decoders, bypass multiplexers, and activating circuits introduce very small energy overheads, and thus are not listed in Table 2.2.

Table 2.2: Operation modes of the ETA cache and the corresponding energy consumption (basic/advanced) per access normalized by the conventional L1 data cache.

	early tag access				actual cache access									
	ETA		conv.		early tag hit		early tag miss only		early TLB miss		re-access		conv	
	access	energy	access	energy	access	energy	access	energy	access	energy	access	energy	access	energy
L1 tag array	0	0	0	0	full	0.01	full	0.01	full	0.01	0	0	full	0.01
L1 data array	0	0	0	0	1	0.257	full/0	0.99/0	full/0	0.99/0	1	0.257	full	0.99
TLB	0	0	0	0	0	0	0	0	full	0.18	0	0	full	0.18
LSQ TLB	full	0.2	0	0	0	0	0	0	0	0	0	0	0	0
LSQ tag array	full	0.012	0	0	0	0	0	0	0	0	0	0	0	0
info buffer	1	0.002	0	0	1	0.002	1	0.002	1	0.002	0	0	0	0
total energy		0.214		0		0.269		1.012/0.012		1.182/0.192		0.257		1.18

2.5.2 Energy Efficiency

Since the proposed ETA cache does not affect the cache miss rate (i.e., no change in the replacement policy for the L1 data cache), the energy consumption related to cache misses, such as data replacement and off-chip memory accesses will be the same as that of the conventional cache. Therefore, we did not include these results in this chapter.

The Simplescalar toolset is employed to collect the statistics of different operations in these simulations. The results of replacements in the LSQ tag arrays and LSQ TLB are also collected. It was shown that the number of early tag and TLB accesses at the LSQ stage is more than that of the actual cache accesses. This is because some memory instructions can get the data directly from the LSQ by store forwarding, and some memory instructions might be flushed from the LSQ thus not issued to the L1 data cache. These results combined with the energy per access shown in Table 2.2 allow us to evaluate different cache configurations.

2.5.2.1 ETA: the basic mode

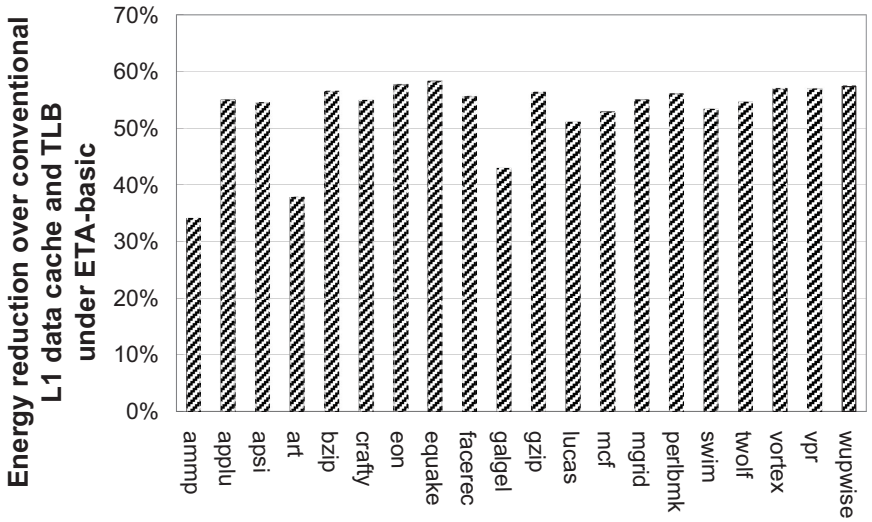


Figure 2.15: Energy reduction of the ETA cache in the basic mode over the conventional L1 data cache and data TLB under SPEC 2000 benchmarks.

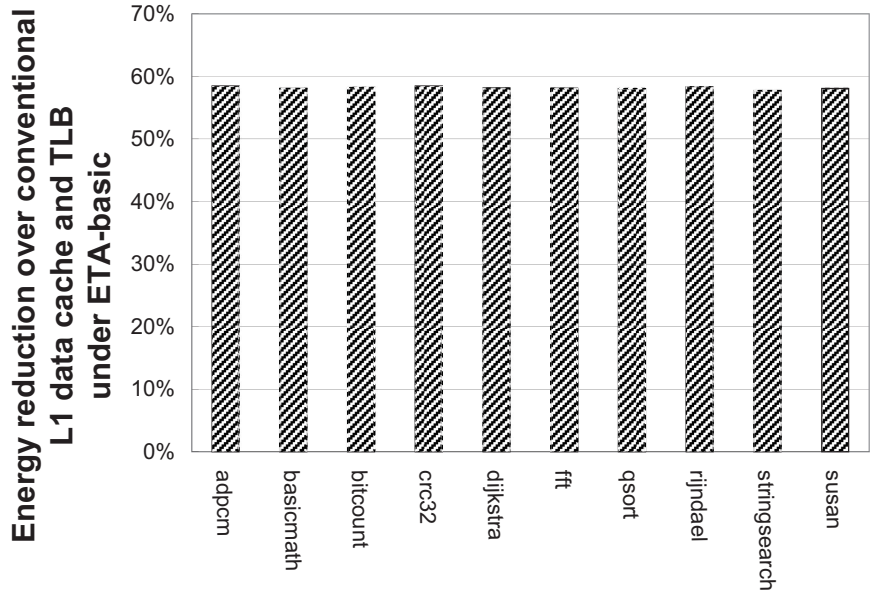


Figure 2.16: Energy reduction of the ETA cache in the basic mode over the conventional L1 data cache and data TLB under MiBench benchmarks.

Fig 2.15 shows the energy reduction achieved by the proposed ETA cache under the

basic mode over the conventional L1 data cache. We observe a 34.1%58.3% energy reduction across different SPEC CPU2000 benchmarks, with an average energy reduction of 52.8%. The energy reduction for the MiBench benchmarks is shown in Fig. 2.16. Due to the higher percentage of early tag determination in the MiBench benchmarks (see Fig. 2.6), the achieved energy reduction is higher than that of the SPEC CPU2000 benchmarks. Note that these results include the energy overheads of early tag and TLB accesses at the LSQ stage, such as the lookup and update operations.

2.5.2.2 ETA: the advanced mode

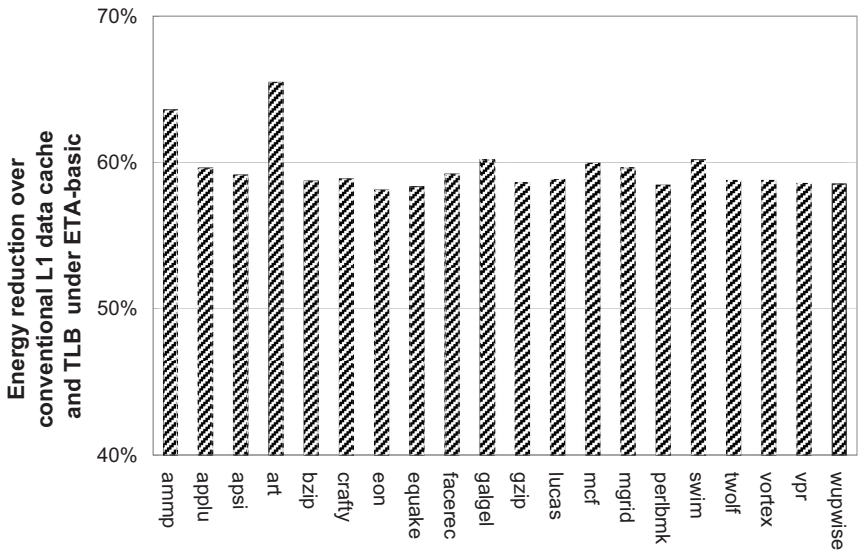


Figure 2.17: Energy reduction of the ETA cache in the advanced mode over the conventional L1 data cache and data TLB under SPEC 2000 benchmarks.

Fig. 2.17 shows the energy reduction of the proposed ETA cache under the advanced mode for the SPEC CPU2000 benchmarks. The energy reduction ranges from 58.4% to 63.6% across different benchmarks with 59.6% on average, all higher than the corresponding measures in the basic mode. The advanced mode is more energy efficient due to fewer ways accessed during the cache access stage. In particular, it is very effective for workloads whose

memory instructions cannot find their early destination ways at the LSQ stage, such as ammp and art. As shown in Fig. 2.17, these two benchmarks achieve 29.3% and 27.7% more energy reduction, respectively, in comparison with the basic mode of the ETA cache.

For MiBench benchmarks, due to their high hit rates of early accesses (see Fig. 2.6), only a small number of memory instructions cannot get the early destination ways at the LSQ stage. Therefore, we do not expect substantially better results in the advanced mode for the MiBench benchmarks. The average energy reduction achieved by the advanced mode is about 1% higher than that of the basic mode.

2.5.2.3 Different Cache Configurations

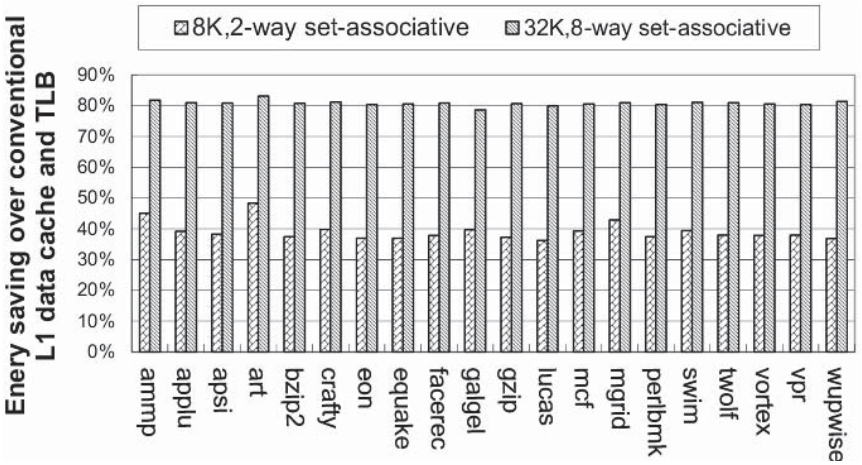


Figure 2.18: Energy reduction of the ETA-advanced cache under different cache configurations.

Fig. 2.18 shows the energy reduction of the ETA-advanced cache for SPEC CPU2000 benchmarks under two different cache configurations: 1) 8 K two-way set-associative, and 2) 32 K eight-way set-associative. Compared with the results in Fig. 2.17, the energy savings under the first configuration are smaller. The reason is that the relative energy overhead of the LSQ tag arrays and LSQ TLB becomes higher as the size of the data cache reduces. Also, as the cache associativity reduces, the energy savings from the data arrays become smaller.

The second cache configuration shows the opposite trend, where larger energy reduction is achieved as compared with the results shown in Fig. 2.17. For MiBench benchmarks, similar results were observed under different cache configurations.

Note that the above energy reduction results were obtained based on total cache accesses. It is true that instructions with different addressing modes have different mechanisms to generate addresses for their cache accesses. However, their cache addresses need to be available during the LSQ stage. As a result, our technique is effective to instructions under different addressing modes as long as they need to access L1 data caches. Also, we reported the energy savings in the L1 data cache only as in this chapter we target L1 data cache exclusively. Smaller energy savings (around 15%) are expected at the entire data cache hierarchy because the L1 data cache only contributes to a portion of the total cache energy.

2.5.3 Design overhead

The design overhead comes mainly from three new components: LSQ TLB, LSQ tag arrays, and information buffer. For the processor under the evaluation, the total size of these components is about 0.8 kB, which is around 2% of the total L1 cache size. The area overhead is approximately in the same range. This overhead is manageable for most embedded processors.

2.6 Conclusions

This chapter presented a new energy-efficient cache design technique for low-power embedded processors. The proposed technique predicts the destination way of a memory instruction at the early LSQ stage. Thus, only one way needed to be accessed during the cache access stage if the prediction is correct, thereby reducing the energy consumption significantly. By applying the idea of phased access to the memory instructions whose early destination ways cannot be determined at the LSQ stage, the energy consumption can be

further reduced with negligible performance degradation. Simulation results demonstrated the effectiveness of the proposed technique as well as the performance impact and design overhead. While our technique was demonstrated by a L1 data cache design, future work is being directed toward extending this technique to other levels of the cache hierarchy and to deal with multithreaded workloads.

Chapter 3

Conclusion and Future Work

3.1 Conclusion

In this thesis, a new temperature aware refresh policy for 3D-stacked DRAM is proposed and a new energy efficient L1 data cache structure is studied. Although the 3D-stacked DRAM has the advantage of high packaging density and short interconnect length, it suffers from much higher power density. The temperature problem is more critical for 3D ICs than it for 2D structure. We focus on an adaptive scheme to solve the challenge of too frequently refreshing in high operation temperature of 3D-stacked DRAM. Exploiting in different angle with most of existing work, this thesis presents a new method to improve DRAM-based memory performance, which is making the refresh rate adjustable based on every memory cell's local temperature. This new method, Temperature Aware Refresh (TAR), is efficient in 3D-stacked DRAM for the reason that it reduces refresh operations for memory cells working in low temperature, meanwhile keeps the high refresh rate for memory cells working in peak temperature. Thereby, the saved time period for DRAM cells is able to issue other memory request, like read and write, instead of too frequently refreshing the exist data. It not only maintains the data in short retention time cells, but also releases more margins for long retention time cells. In contrast with the conventional performance improving principle for DRAM, our work introduces a new way for DRAM performance improving, which is applying various refresh rates for each DRAM bank.

Though some existing works have shown that 3D DRAM has the potential to be used as cache due to its high packaging density and good performance, problems still exist making it hard to implement. However, the conventional cache architecture still has chances to be improved for better performance and energy efficiency. In this thesis we present a new energy-efficient cache design technique for low-power embedded processors, which predicts the destination way of a memory instruction at the early LSQ stage. Inspired by the idea

of phased access to the memory instructions, we propose a new structure for L1 data cache that enables only one way to be accessed during the cache access stage, if the prediction is correct. While there are existing some performance impacts and design overheads, the power consumption of L1 cache is saved a lot, which is demonstrated by simulation results.

3.2 Future Work

The adaptive schemes shown in this thesis are only two examples of our work being towards the energy efficiency and performance improvement IC design. The work presented in this thesis is relatively simple and easy to follow, though; we introduce two novel ways for improving performance in 3D-stacked DRAM and saving energy in conventional cache design.

The thermal condition of 3D IC is very researchable. A better sensor location will lead to less number of temperature sensors so that there will be less design overhead and the refresh rate will be more accurate for each bank. In other aspect, TAR is able to be combined with most existing works, like subarray-level parallelism. In some "ture" 3D DRAM structure, each bank is divided into subarray-based structure which different subarrays are in different layers. 3D DRAM is a new technique that requires a lot of researches to implement into our daily life. For energy efficient cache design, the performance impaction and hardware overhead should be further reduced. In addition, while our technique was demonstrated by a L1 data cache design, future work is being directed toward extending this technique to other levels of the cache hierarchy and to deal with multi-threaded workloads. What's more, the 3D-DRAM cache design is a very bright research field for its high density and better performance. The problem for 3D-DRAM cache right now is how to solve the refresh performance impaction in high speed read and write requests condition.

References

[1] G.L. Loi, B. Agrawal, N. Srivastava, S.C. Lin, T. Sherwood, and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy", In *Proceedings of the 43rd annual Design Automation Conference,* ACM, July 2006, pp. 991-996.

[2] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G.H. Loh, ..., and C. Webb, "Die Stacking (3D) Microarchitecture", In *Microarchitecture, 2006. MICRO-39. IEEE 39th Annual IEEE/ACM International Symposium on,* IEEE, December 2006, pp. 469-479.

[3] G.H. Loh, "3D-Stacked Memory Architectures for Multi-core Processors", In *Computer Architecture, 2008. ISCA '08. 35th International Symposium on,* IEEE, June 2008, pp. 453-464.

[4] H. Sun, J. Liu, R.S. Anigundi, N. Zhang, J.Q. Lu, K. Rose, and T. Zhang, "3D DRAM Design and Application to 3D Multicore Systems", In *IEEE Design & Test of Computers 26(5),* IEEE, Oct 2009, pp. 36-47.

[5] Tezzaron Semiconductors, "3D Stacked DRAM/Bi-STAR Overview", `http://www.tachyonsemi.com/memory/Overview\_3D\_DRAM.htm`, 2008,

[6] D. Zhao, H. Homayoun, and A.V. Veidenbaum, "Temperature Aware Thread Migration in 3D Architecture with Stacked DRAM", In *Quality Electronic Design (ISQED), 2013 14th International Symposium on,* IEEE, March 2013, pp. 80-87.

[7] JEDEC, "DDR3 SDRAM Standard", 2010.

[8] ITRS, "International technology roadmap for semiconductors executive summary", `http://www.itrs.net/Links/2011ITRS/2011Chapters/2011ExecSum.pdf`, 2011, p.83.

[9] M. Shevgoor, J.S. Kim, N. Chatterjee, R. Balasubramonian, A. Davis, and A.N. Udipi, "Quantifying the Relationship between the Power Delivery Network and Architectural Policies in a 3D-Stacked Memory Device", In *MICRO,* ACM, 2013, pp. 198-209.

[10] A. Annamalai, R. Kumar, A. Vijayakumar, and S. Kundu, "A System-level Solution for Managing Spatial Temperature Gradients in Thinned 3D ICs", In *Quality Electronic Design (ISQED), 2013 14th International Symposium on,* IEEE, March 2013, pp. 88-95.

[11] J. Stuecheli, D. Kaseridis, H.C. Hunter, and L.K. John, "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory", In *Computer Architecture (ISCA), 2012 39th Annual International Symposium on,* IEEE, June 2012, pp. 1-12.

[12] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM", In *Computer Architecture (ISCA), 2012 IEEE 39th Annual International Symposium on,* IEEE, June 2012, pp. 368-379.

[13] P. Nair, C.C. Chou, and M.K. Qureshi, "A Case for Refresh Pausing in DRAM Memory Systems", In *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on,* IEEE, February 2013, pp. 627-638.

[14] M. Ghosh, and H.H.S Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs", In *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on,* IEEE Computer Society, December 2007, pp. 134-145.

[15] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh", In *Computer Architecture (ISCA), 2012 IEEE 39th Annual International Symposium on,* IEEE, June 2012, pp. 1-12.

[16] T.V. Kalyan, K. Ravi, and M. Mutyam, "Scattered Refresh: An Alternative Refresh Mechanism to Reduce Refresh Cycle Time", In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific,* IEEE, Jan 2014, pp. 598-603.

[17] K.K.W. Chang, D. Lee, Z. Chishti, A.R. Alameldeen, C. Wilkerson, Y. Kim, and O. Mutlu, "Improving DRAM Performance by Parallelizing Refreshes with Accesses", In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th International Symposium on,* IEEE, Feb 2014, pp. 356-367.

[18] M. Sadri, M. Jung, C. Weis, N. Wehn, and L. Benini, "Energy Optimization in 3D MP-SoCs with Wide-I/O DRAM Using Temperature Variation Aware Bank-wise Refresh", In *Proceedings of the conference on Design, Automation & Test in Europe(DATE),* European Design and Automation Association, March 2014 , p. 281.

[19] MICRON, "4Gb: x4, x8, x16 DDR3L SDRAM", 2013

[20] J.Y. Sim, H. Yoon, K.C. Chun et al., "A 1.8-V 128-Mb mobile DRAM with double boosting pump, hybrid current sense amplifier, and dual-referenced adjustment scheme for temperature sensor", In *Solid-State Circuits, IEEE Journal of 38(4),* IEEE, Apr 2003, pp 631-640.

[21] C.K. Kim, B.S. Kong, C.G. Lee, Y.H. Jun, "CMOS Temperature Sensor with Ring Oscillator for Mobile DRAM Self-refresh Control", In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, IEEE, May 2008, pp 3094-3097.

[22] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, ..., and D.A. Wood, "The gem5 simulator", In *ACM SIGARCH Computer Architecture News, 39(2),* 2011, pp. 1-7.

[23] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator", In *Computer Architecture Letters 10(1),* IEEE, March 2011, pp. 16-19.

[24] JEDEC, "Low Power Double Data Rate 3 (LPDDR3)", 2013.

[25] MICRON, "2Gb: x16, x32 Automotive LPDDR2 SDRAM", 2014

[26] SPEC CPU2006, "Standard Performance Evaluation Corporation", `http://www.spec.org/cpu2006`

[27] Intel. Intel XScale Microarchitecture, 2001.

[28] C. Zhang, Frank Vahid, and Walid Najjar "A highly-configurable cache architecture for embedded systems," *Intl. Symp. on Computer Architecture*, pp.136-146, 2003.

[29] S. Segars, "Low power design techniques for microprocessors," *International Solid-State Circuits Conference Tutorial*, 2001.

[30] S. Manne, A. Klauser, and D. Grunwald, "Pipline gating: spculation conrol for energy reduction," *International Symposium on Computer Architecture,* pp 132-141, 1998.

[31] M. Gowan, L. Biro, and D. Jackson, "Power considerations in the design of the alpha 21264 microprocessor," *Design Automation Conference*, 1998.

[32] A. Malik, B. Moyer and D. Cermak, "A Low power unified cache architecture providing power and performance flexibility," *Intl. Symp. on Low Power Electronics and Design*, pp. 241-243, 2000.

[33] T. Lyon, E. Delano, C. McNairy and D. Mulla, "Data Cache Design Considerations for the Itanium Processor," *IEEE Conference on Computer Design*, pp. 356-362, 2002.

[34] D Nicolaescu, A. Veidenbaum, and A. Nicolau, "Reducing power consumption for high-associativity data caches in embedded processors," *Design, Automation, and Test in Europe*, pp. 1064-1068, 2003.

[35] Chuanjun Zhang et al., "A way-halting cache for low-energy high-performance systems," *International Symposium on Low Power Electronics and Design*, pp. 126-131, 2004.

[36] J. Montanaro et al., "A 160-MHz 32-b 0.5-V CMOS RISC microprocessor," *IEEE JSSC, 31(11)* pp. 1703-1714, 1996.

[37] S. Santhanam et al., " A low-cost, 300-MHz, RISC CPU with attached media processor," *IEEE JSSC, 33(11)* pp. 1829-1838, 1998.

[38] D. Brooks, et.al., "Wattch: a framework for architectural-level power analysis and optimizations," *International Symposium on Computer Architecture*, pp. 83-94, 2000.

[39] A. Hasegawa, I. Kawasaki, K. Yamada, S. Yoshioka, S. Kawasaki, and P. Biswas, "Sh3: High code density, low power," *IEEE Micro*, pp. 11-19, 1995.

[40] J. Dai and L. Wang, "An energy-efficient L2 cache architecture using way tag information under write-through policy," *IEEE Trans. on VLSI Systems*, accepted.

[41] M. Powell, A. Agarwal, T. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," *MICRO*, pp. 54 - 65, 2001.

[42] B. Calder and D. Grunwald, "Next cache line and set prediction," *IEEE Symposium on High-Performance Computer Architecture*, pp. 49 - 60, 1996.

[43] B. Batson and T. Vijaykumar, "Reactive associative caches," *International Conference on Parallel Architectures and Compiliation*, pp. 82 - 92, 2001.

[44] T. M. Austin and G. Sohi, "Zero-cycle loads: microarchitecture support for reducing load latency," *International Symposium on Microarchitecture*, 1995.

[45] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," *Intl. Symp. on Low Power Electronics and Design*, pp. 273-275, 1999.

[46] A. Ma, M. Zhang, and K. Asanovi, "Way memoization to reduce fetch energy in instruction caches," *ISCA Workshop on Complexity Effective Design*, pp. 1-9, 2001.

[47] T. Ishihara and F. Fallah, "A Way Memoization Technique for Reducing Power Consumption of Caches in Application Specific Integrated Processors," *Proc. of Design Automation and Test in Europe Conference*, pp. 358-363, 2005.

[48] M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, and K. Roy, "Reducing set-associative cache energy via way-prediction and selective direct-mapping," *International Symposium on Microarchitecture*, pp. 54-65, 2001.

[49] J. Li and Y. Hwang, "Snug set-associative cache: reducing leakage power while improving performance," *International Symposium on Low Power Electronics and Design*, pp. 345-350, 2005.

[50] D. Nicolaescu, A. Veidenbaum, and A. Nicolau, "Reducing data cache energy consumption via cached load/store queue," *International Symposium on Low Power Electronics and Design*, pp. 252-257, 2003.

[51] L. Jin and S. Cho, "Reducing cache traffic and energy with macro data load," *International Symposium on Low Power Electronics and Design*, pp. 147-150, 2006.

[52] T. Austin, E. Larson, and D. Ernst. "Simplescalar: an infrastructure for computer system modeling," *IEEE Com- puter*, 35(2), 2002.

[53] E. Delano and D. Mulla, "Data cache design considerations for the Itanium2 processor," *Intl. Conf. on Computer Design*, pp. 356-362, 2002.

[54] B. Brock and M. Exerman," Cache latencies of the PowerPC MPC7451," *Freescale Semiconductor Application Note*, Rev. 2, 2006.

[55] SPEC CPU2000 at *http://www.spec.org/cpu*

[56] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *IEEE 4th Annaul Workshop on Workload Characterization*, 2001.

[57] L. Hennessey and D.A. Patterson, "Computer architecture: a quantitative approach. 4th Edition," *Elsevier Science & Technology Books*, 2006.

[58] R. Min, Z. Xu, Y. Hu, and W. Jone, "Partial tag comparison: a new technology for power efficient set-associative cache designs," *Intl. Conf. on VLSI Design*, pp. 183-188, 2004.

[59] CACTI 5.3 at *http://www.hpl.hp.com/research/cacti/*