

國立臺灣大學電機資訊學院資訊工程學系

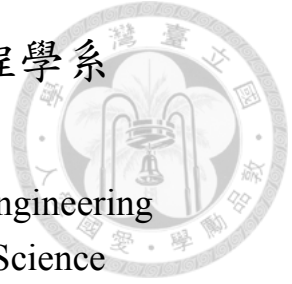
博士論文

Department of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Doctoral Dissertation



多核心系統中動態隨機存取記憶體之低功率設計及溫度控制

A Low-Power DRAM System with Thermal Control for

Multi-Core Systems

林仲祥

Chung-Hsiang Lin

指導教授：楊佳玲博士

Advisor: Chia-Lin Yang, Ph.D.

中華民國 103 年 7 月

July, 2014

中文摘要



在現今的電腦系統中，動態隨機存取記憶體 (DRAM) 常被用作主記憶體，但其效能已漸漸跟不上處理器，此一問題在多核系統中將因多個處理核心同時存取記憶體而日益嚴重。為了要滿足多核系統的需求，記憶體容量與頻寬都需隨之增加。現今的研究亦發現記憶體的功耗與溫度也越來越高，因此在多核系統中，記憶體系統設計的挑戰是在降低其功耗的同時，滿足效能與工作溫度的限制。由於背景功耗 (background power) 佔記憶體功耗的一半以上，在此論文中，我將專注於減少記憶體的背景功耗，並避免降低效能且同時滿足工作溫度的限制。記憶體背景功耗由週邊電路漏電流 (peripheral leakage) 及刷新功耗 (refresh power) 組成。為減少週邊電路漏電流，我提出效能/功耗/溫度共同管理架構 (joint Performance, Power and Thermal management framework, PPT framework)，透過工作排程與分頁配置，在不同的系統頻寬需求下，滿足效能與工作溫度的限制並降低記憶體功耗。為減少刷新功耗，我提出刷新節能選擇性錯誤更正架構 (Selective Error Correction for Refresh Energy reduction framework, SECRET framework)，透過延長刷新間隔 (refresh interval) 來減少刷新功耗，並將延長刷新間隔時產生的記憶錯誤 (retention error) 視為硬體錯誤 (hard error)，只為這些少數會發生錯誤的記憶體單元配置錯誤更正資訊，以減少錯誤更正機制的額外成本。由於上述兩個架構的設計並無衝突，且用完全獨立的方式減少不同種類的功耗，因此兩者可以同時使用，以進一步達到節能省電的效果。

關鍵字 - 動態隨機存取記憶體、功率、溫度、週邊電路、刷新

Abstract



DRAMs are used as the main memory in most computing systems today. However, memory system performance has been historically lagged behind CPU performance, and this problem exacerbates in a multi-core system since memory resources are shared by multiple cores on a chip. To sustain concurrent memory requests from multiple cores, the speed, bandwidth, and capacity of DRAM memories continue to increase. Studies show that DRAMs now consume a significant part of the overall system power, and the temperature of DRAMs is also approaching its limit. Therefore, the challenge that we are facing today from DRAM memory management is how to achieve desired DRAM power reduction, and meet the performance and thermal constraints at the same time. Since the background power of DRAMs usually consumes more than 50% of the total DRAM power, I focus on reducing DRAM background power with negligible performance overhead, and meeting the DRAM thermal constraint in this dissertation.

The background power of DRAMs is composed of power consumption of peripheral leakage which depends on DRAM power states and refresh power. To reduce the DRAM peripheral leakage, I propose a joint performance, power and thermal management framework (PPT) through orchestrating task execution and page allocation to exploit DRAM low-power modes efficiently. The PPT framework adapts to system loading to maximize peripheral leakage power savings and avoid memory thermal hotspot at the same time while sustaining the system bandwidth demand. For refresh power reduction, I propose SECRET (Selective Error Correction for Refresh Energy reduction) that is designed to reduce the inevitable refresh processes by prolonging the refresh interval and correcting the retention errors by ECC (Error Correcting Code). The key observation I make is that retention errors can be

treated as hard errors rather than soft errors, and only few DRAM cells have large leakage to cause retention errors. Therefore, instead of equipping error correction capability in all memory cells as existing ECC schemes, I only allocate error correction information to leaky cells under a refresh interval to minimize the overheads of ECC.

The architectural supports for these two techniques do not conflict, so they can be used at the same time. Since both techniques incur negligible performance degradation, adopting them together would only hurt performance slightly as well. The effectiveness for power reduction and thermal control of these two techniques used simultaneously is as good as that of these two techniques used separately, because they reduce different parts of the DRAM background power and there is no interference between these two methods. So, utilizing the PPT and SECRET frameworks can reduce both the peripheral leakage power and refresh power of DRAM systems and alleviate the operating temperature with negligible overheads in performance and hardware modifications.

keywords - DRAM, Power, Thermal, Peripheral Leakage, Refresh

Table of Contents

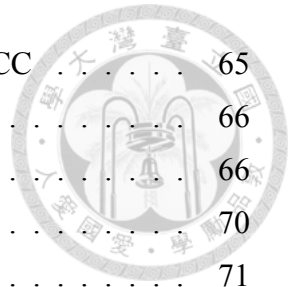


中文摘要	i
Abstract	ii
List of Figures	vii
List of Tables	ix
Chapter 1. Introduction	1
Chapter 2. Related Work	5
2.1 Peripheral Leakage Reduction	5
2.1.1 Idle Period Prolongation	5
2.1.1.1 Power-Aware Data Allocation	6
2.1.1.2 Temporal Alignment of Memory Requests	8
2.1.1.3 Data Re-computation	10
2.1.1.4 Thermal Drawback of Request Clustering	10
2.1.2 Power Mode Decision	10
2.2 Refresh Power Reduction	13
2.2.1 Refresh Reduction by Sensing Leakage Current	14
2.2.2 Refresh Reduction by Considering Access Pattern	15
2.2.3 Refresh Reduction by Considering Data Property	16
2.2.4 Multi-Period Refresh	19
2.2.5 ECC for Refresh Reduction	20
2.2.6 Adaptive Refresh Interval	22
2.3 Dynamic Thermal Management for DRAMs	24
2.4 Reliable Low-Voltage Operation for SRAM Cache	26
2.5 Related Work Using Ideas Similar to the PPT Framework	29



Chapter 3. Introduction to DDRx-SDRAM	31
3.1 DRAM Organization	31
3.2 Power States of DDRx-SDRAM	32
3.3 DRAM Refresh Operation	32
Chapter 4. PPT Framework: DRAM Peripheral Leakage Reduction with Thermal Control	35
4.1 Introduction	35
4.2 The PPT Framework	36
4.2.1 Adaptive Grouping Mechanism	37
4.2.1.1 PPT Configuration Selection	38
4.2.1.1.1 Static Bandwidth (BW) Estimation Method	38
4.2.1.1.2 Dynamic Bandwidth (BW) Estimation Method	40
4.2.1.2 New PPT Configuration Adoption	42
4.2.1.3 Architectural Support	44
4.2.2 Thermal Control Policy	45
4.3 Evaluation to PPT Framework	46
4.3.1 Experimental Setup	46
4.3.2 Experimental Results	49
4.3.2.1 Power/Performance Evaluation of Adaptive Grouping	49
4.3.2.2 Deferred Page Migration Analysis	56
4.3.2.3 Power State Analysis	58
4.3.2.4 Thermal Evaluation	59
4.3.2.5 Summaries of Performance, Power and Temperature of PPT	61
Chapter 5. SECRET Framework: DRAM Refresh Power Reduction	63
5.1 Introduction	63
5.2 The SECRET Framework	64
5.2.1 Main Idea	64
5.2.2 Candidates of the Error Correcting Scheme in SECRET	65
5.2.2.1 Hamming Code ECC	65

5.2.2.2	Bose-Chaudhuri-Hocquenghem (BCH) Code ECC	65
5.2.2.3	Error Correcting Pointer	66
5.2.3	Selective Error Correction (SEC)	66
5.2.4	Off-line Phase: ECP Directory/ECPs Construction	70
5.2.5	Refresh Interval Adaptation	71
5.2.5.1	Profiling Memory Cells for Refresh Interval Adaptation	72
5.2.5.2	Deduction of the Worst Case Leakage Ratio	73
5.2.5.3	Proof of the Correctness of Refresh Interval Adaptation	74
5.2.6	Discussion on Overheads of SEC	76
5.3	Evaluation to SECRET Framework	77
5.3.1	Experimental Setup	77
5.3.2	Experimental Results	79
5.3.2.1	Design Space Exploration: Deciding Target Error Rate and SEC Cache Configuration	80
5.3.2.2	Energy Analysis	82
5.3.2.3	Performance Analysis	84
5.3.2.4	Evaluating SECRET with Reduced Last-level Cache Size	85
5.3.2.5	Evaluation of Distributing Leaky Cells with Spatial Locality	87
5.3.2.6	Comparison with Traditional ECC Approaches	89
Chapter 6. Conclusion		91
Bibliography		95

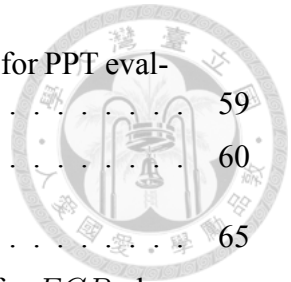




List of Figures

3.1	The structure of a DRAM cell and a 2D DRAM array.	32
3.2	The power state transition of DDRx-SDRAM.	33
3.3	Data retention time distribution of DRAM cells.	33
4.1	4-group PPT configuration.	36
4.2	Group scheduling of the PPT framework.	37
4.3	Static BW estimation method for Adaptive Grouping in the PPT framework.	39
4.4	Dynamic BW estimation method for Adaptive Grouping in the PPT framework.	41
4.5	1-group PPT configuration.	46
4.6	Memory throughput breakdown of threads in SPEC2000-1 with the performance-driven policy for PPT evaluation.	50
4.7	Throughput and power of DRAM system with SPEC2000-1 for PPT evaluation.	51
4.8	The throughput of the performance-driven policy, the number of ranks per group for the static and dynamic BW estimation methods, and the number of concurrently accessed ranks for the power-driven policy in SPEC2000-1 for PPT evaluation.	52
4.9	Memory throughput breakdown of threads in SPEC2000-2 with the performance-driven policy for PPT evaluation.	53
4.10	Throughput and power of DRAM system with SPEC2000-2 for PPT evaluation.	54
4.11	Memory throughput breakdown of threads in SPECjbb with the performance-driven policy for PPT evaluation.	55
4.12	Throughput and power of DRAM system with SPECjbb for PPT evaluation.	56
4.13	Rank Miss Rate, Hot/Cold Access Rate, New Page Rate and the number of page migrations for SPEC2000-1 with the dynamic BW estimation method in 0.5s~1.0s and 4.6s~5.1s for evaluations to Deferred Page Migration.	57

4.14	Power state breakdown of DRAM system with SPEC2000-1 for PPT evaluation.	59
4.15	Temperature of DRAM system for PPT evaluation.	60
5.1	SECRET framework.	65
5.2	(a) Error correcting pointer. (b) Hardware implementation for ECP_1 decoder [1].	67
5.3	Error correcting information for the SEC mechanism.	68
5.4	Architecture of the SEC cache.	68
5.5	System architecture of the SEC mechanism.	70
5.6	Data retention time distribution of DRAM cells and bits in Set 1 and Set 2.	73
5.7	Retention error rates of utilizing various refresh intervals.	79
5.8	Refresh power reduction achieved by utilizing refresh intervals of various target retention error rates.	81
5.9	Average cache miss rate of SEC cache with varied number of ways and sets.	81
5.10	SEC cache line size vs. DRAM power reduction under various retention error rates.	82
5.11	Power consumption of the DRAM system with SECRET normalized to the baseline.	83
5.12	Power breakdown of DRAM peripheral leakage, dynamic and refresh power of the baseline (left) and SECRET (right).	84
5.13	Additional memory accesses of the SECRET framework normalized to the number of data accesses issued by the workloads.	85
5.14	DRAM system power breakdown normalized to the DRAM power consumption with 8MB L2 cache.	86
5.15	Average additional memory accesses of the SECRET framework normalized to the number of data accesses issued by the workloads with 8MB, 4MB and 2MB L2 cache.	87
5.16	DRAM power reduction of various distributions, and the maximum number of retention errors in a region with the target error rate set to 10^{-6}	89
5.17	Mean time to failure vs. retention error rate in the DRAM system with Hamming Code ECC.	89





List of Tables

4.1	Processor and memory configurations for PPT evaluation.	47
4.2	Workloads for PPT evaluation.	48
4.3	Normalized throughput, power and peak temperature of the performance-driven policy, the power-driven policy and the PPT mechanisms.	61
5.1	Number of bits/bytes of each ECP directory/ECP field in the SECRET framework.	76
5.2	System configurations for SECRET evaluation.	78
5.3	Workloads for SECRET evaluation.	78



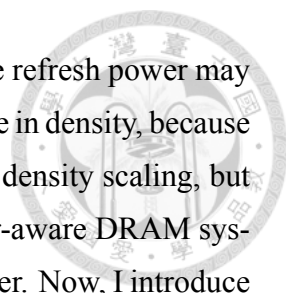


Chapter 1

Introduction

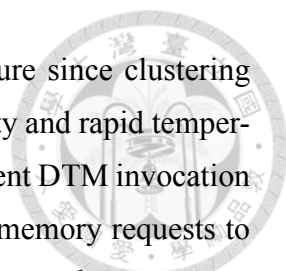
Optimizing power consumption has become a critical design issue not just for battery-operated mobile devices, but also for high end systems due to the reliability issue and cooling/packaging cost. To achieve an energy-efficient design, all system components need to be considered. DRAMs are used as the main memory in most computing systems today. Studies show that DRAMs consume over 30% system power in handheld computers [2], up to 40% system power in commercial servers [3], and about 20% system power in mobile phones [4, 5]. The memory system is obviously one of the main contributors to the overall system power consumption. Recently, multi-core processors are widely adopted from smart phones to servers. To sustain the increasing bandwidth demand in multi-core systems, the capacity and speed of DRAM memories are also expected to grow significantly. This will lead to increasing power consumption in the memory system. Therefore, it is ever increasingly important to design a power-aware memory system.

The DRAM power consumption falls into three different components: background power, activation power and read/write burst power. A previous study shows that the background power usually consumes more than 50% of the total DRAM power [6]. So, I focus on reducing the background power of DRAMs in this dissertation. The background power includes peripheral leakage power and refresh power which are both consumed due to leakage currents in DRAMs. The peripheral leakage power comes from the leakage power of DRAM peripheral circuits that include row/column decoders, sense amplifiers, row buffers, etc. When the peripheral circuits are turned on, the peripheral leakage power is a main contributor of the DRAM power consumption, but the peripheral leakage power can be reduced by turning the peripheral circuits off. The refresh power is generated by refresh operations that are required to recharge the capacitors in DRAMs to avoid data loss



due to leakage currents of DRAM cells. A recent study shows that the refresh power may become the dominant component of DRAM power when DRAMs scale in density, because other components of DRAM power increase slowly with the DRAM density scaling, but the refresh power increases linearly [7]. Therefore, to design a power-aware DRAM system, I focus on reducing the peripheral leakage power and refresh power. Now, I introduce the proposed techniques.

For reducing the DRAM peripheral leakage power, I propose a management framework that takes performance, power and thermal all into accounts. The most common approach to reduce the DRAM peripheral leakage power is to utilize the inactive low-power modes that turn some components of peripheral circuits off when the DRAM modules are idle. Since the DRAM modules can not serve memory requests in the inactive low-power modes, to improve the efficiency of the inactive low-power modes, there are two kinds of mechanisms proposed to prolong the idle periods of DRAM modules. The power-aware data allocations [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25] cluster data accesses in a small number of memory modules and put the remainings into a low-power mode to reduce peripheral leakage. Although popularity data layout is good for power, it is not able to expose memory parallelism thereby impairing memory system performance. On the other hand, the mechanisms of memory request alignment [26, 27, 28, 29, 30, 31, 23] cluster memory requests in a burst to prolong the idle periods of DRAM modules. In the long idle periods, the inactive low-power modes can be used efficiently, but memory request alignment may incur performance degradation due to delaying the requests. In contrast to power reduction, to improve the performance of DRAM systems, some purely performance-driven memory systems use data striping to explore maximum parallelism among memory accesses to gain performance [32, 33, 34, 35, 36, 37, 38, 39, 40]. However, activating all memory modules at the same time reduces the opportunities of utilizing inactive low-power modes, so data striping usually gains performance at the expense of high power consumption. Besides the tradeoff between performance and power, due to the high power consumption of the memory system, memory thermal management is becoming a critical issue recently [41, 42]. To protect the memory system from thermal emergency, several dynamic thermal management (DTM) techniques have been proposed [42, 43, 44, 45, 46, 47, 48, 49, 25]. Although reducing peripheral leakage power by aforementioned low-power techniques in general can lower



average temperature, it may cause adverse effect on peak temperature since clustering memory requests in few modules in a burst causing high power density and rapid temperature rise in active modules. This may consequently incur more frequent DTM invocation with performance degradation. The thermal drawback of clustering memory requests to reduce peripheral leakage is discussed in detail in Section 2.1.1.4. Because there are complex interactions between performance, power and thermal factors, these three factors all need to be taken into account in designing a power-aware memory subsystem for reducing peripheral leakage. However, most of previous works look at one or two factors only. Due to the interplay among performance, power and thermal, a technique optimized for one factor only often causes uncontrollable degradation at other design metrics. To tackle this issue, in this dissertation, I propose the first joint performance, power and thermal (PPT) management framework for DRAM memory in multi-core systems. The details of the PPT framework are described in Chapter 4.

For DRAM refresh power reduction, I propose a novel error correction framework for retention errors in DRAMs. Due to process variation, memory cells exhibit retention time variations. However, current DRAMs use a single worst-case refresh period. Since prolonging refresh intervals introduces retention errors, previous works adopt conventional ECC (Error Correcting Code) to correct retention errors. These approaches introduce significant area and energy overheads. For example, in a conventional (72, 64) Hamming code, eight DRAM chips are paired with an extra chip, which requires 12.5% area overhead and additional power consumption. In this dissertation, I propose a novel error correction framework for retention errors in DRAMs, called SECRET (Selective Error Correction for Refresh Energy reduction). The key observation I make is that retention errors can be treated as hard errors rather than soft errors, and only few DRAM cells have large leakage. Therefore, instead of equipping error correction capability in all memory cells as existing ECC schemes, I only allocate error correction information to leaky cells under a refresh interval. My SECRET framework contains two parts, an off-line phase to identify memory cells with retention errors given a target error rate, and a low-overhead error correction mechanism. The experimental results show that the proposed SECRET framework can reduce significant DRAM refresh power with negligible area and performance overheads. The details of the SECRET framework are described in Chapter 5.

The rest of this dissertation is organized as follows. The discussion of the related works is presented in Chapter 2. Chapter 3 shows the basics of DDRx-SDRAM. Chapter 4 explains the proposed joint performance, power, thermal management (PPT) that reduces DRAM peripheral leakage with thermal control. Chapter 5 shows the proposed SECRET framework that reduces DRAM refresh power. Conclusions are given in Chapter 6.



Chapter 2

Related Work

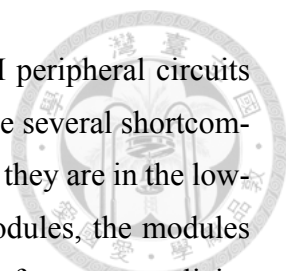
In this chapter, I first introduce the related works about reducing DRAM background power. The studies that focus on reducing peripheral leakage power of DRAMs are discussed in Section 2.1, and the mechanisms which reduce refresh power are mentioned in Section 2.2. Then, the dynamic thermal management policies for DRAMs are presented in Section 2.3. Since the design issues for building a reliable SRAM cache under low supply voltage are similar to that for reducing DRAM refresh power, I also introduce the studies that develop reliable low-voltage operations for SRAM caches in Section 2.4. In the end, I show some works that present some techniques similar to the PPT framework in Section 2.5.

2.1 Peripheral Leakage Reduction

Traditionally, architects usually use inactive low-power modes of DRAM modules to reduce the peripheral leakage. However, DRAM modules can not serve requests in these low-power modes. So, to maximize the efficiency of the low-power modes and minimize performance degradation due to the latency of re-activation for serving requests, it is quite important to create long idle periods and select appropriate low-power modes for DRAM modules in the idle periods. In this section, I introduce previous studies that prolong the idle periods of DRAM modules in Section 2.1.1, and present the works that discuss how to select the appropriate low-power mode in Section 2.1.2.

2.1.1 Idle Period Prolongation

The most promising approach to reduce the peripheral leakage power of DRAMs is to utilize the low-power modes of DRAM modules. In the low-power modes, to re-

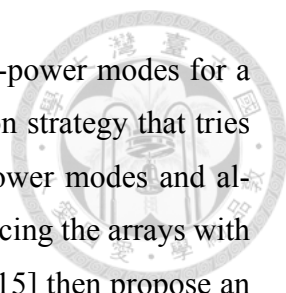


duce the peripheral leakage power, some components of the DRAM peripheral circuits are turned off. However, conventional DRAM low-power modes have several shortcomings. First, the DRAM modules can not serve memory requests when they are in the low-power modes. When there are read requests sent to the memory modules, the modules must be transited back to the active mode to serve the requests. Therefore, most policies only transit the DRAM modules into low-power modes when they are idle to minimize the performance degradation due to high latency for bringing the DRAM modules back to the active mode. Second, transiting the DRAM modules between the active mode and the low-power modes may incur power overheads. So, frequent transitions between the active mode and low-power modes due to short idle periods may cause adverse effect for power reduction. To overcome these two limitations, many works focus on prolonging the idle periods of memory modules to minimize performance and power overheads of power state transitions and maximize the power savings from using low-power modes for the memory subsystems.

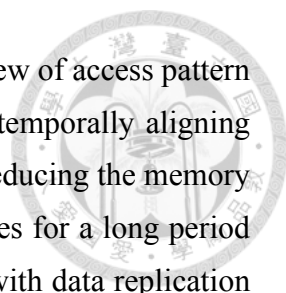
2.1.1.1 Power-Aware Data Allocation

The policies of power-aware data allocation for prolonging the idle periods of memory modules mainly focus on clustering the data with similar access patterns into a subset of memory modules, and transiting the other memory modules into low-power modes when these data are being accessed.

Delaluz et al. [9, 8, 11, 12] introduce a compiler-based optimization framework to cluster the arrays that have similar patterns of lifetime into the same set of memory modules and transit these modules into low-power modes when the lifetime of these arrays ends. Lebeck et al. [10] propose the sequential first-touch page allocation policy that is implemented in the operating system. This policy allocates pages in the order they are accessed and fills an entire DRAM module before moving onto the next. So, this policy minimizes the number of DRAM modules utilized for applications, and places other unused DRAM modules in the low-power modes to reduce energy consumption. Athavale et al. [13] use data layout transformations to change the storage order, improve spatial locality of arrays and their cache behavior, and confine the memory requests to a particular DRAM module for a period of time. The data layout transformations can increase the average inter-access times between two memory requests to the same DRAM module



to provide an opportunity for transiting the DRAM module into low-power modes for a longer time. Delaluz et al. [14] describe an automatic data migration strategy that tries to increase the number of banks which can be transited into low-power modes and allow the use of more aggressive low-power modes by dynamically placing the arrays with temporal affinity into the same set of DRAM banks. Delaluz et al. [15] then propose an array interleaving mechanism which minimizes the number of DRAM modules that need to be active at a given time to put other DRAM modules into low-power modes by clustering the data elements of multiple arrays that are accessed simultaneously into a single common data space. Tuck et al. [16] introduce a frequency-based dynamic page placement policy which is implemented in the operating system. This policy tries to allocate the most frequently accessed pages in the same DRAM modules to provide opportunities to transit other DRAM modules into low-power modes. Huang et al. [17] propose another page allocation policy that minimizes the total number of active DRAM modules used per process. Then, the low-power mechanism activates the DRAM modules used by a process when that process is scheduled, and leaves other DRAM modules in the low-power modes to save energy. Wang et al. [18, 19] introduce a novel memory access graph model to capture potential energy savings and potential performance improvement simultaneously by variable partitioning and memory operation scheduling in the multi-bank memory system with multiple memory operating modes. Huang et al. [20] perform page migration to allocate frequently-accessed pages on hot memory ranks, and infrequently-used and unmapped pages on cold ranks to elongate the average inter-arrival time of memory requests by almost 2 orders of magnitude on cold ranks, where the ultra-low power self-refresh mode can be more utilized. Ozturk et al. [21] combine data migration and data compression to reduce memory power consumption. Data migration puts the data blocks with similar access patterns/lifetimes into the same set of DRAM modules to increase the chances for utilizing low-power modes, and data compression squeezes the data blocks and makes it possible to increase the number of inactive DRAM modules that can be transited into low-power modes. Ozturk et al. [22] then discuss the effectiveness of employing nonuniform bank sizes for reducing memory energy consumption. They propose an integer linear programming based approach that provides the optimal nonuniform bank sizes with data-to-bank mapping, and study a data migration scheme to further improve power savings over nonuniform banking. Pandey et al. [23] cluster frequently

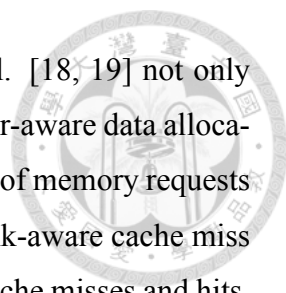


accessed pages in a small subset of DRAM modules to exploit the skew of access pattern in workloads. The data layout can increase the energy savings by temporally aligning the memory requests directed towards the hot DRAM modules and reducing the memory requests to cold DRAM modules that would stay in low-power modes for a long period of time. Ozturk et al. [50] propose a novel data allocation strategy with data replication for multi-bank memory systems. This strategy puts the data blocks with similar access pattern in the same DRAM modules, and replicates the data blocks when necessary to prevent re-activating idle DRAM modules that are in the low-power modes. Kumar et al. [24] restrict the amount of DRAM modules that are available to applications to tradeoff performance and power. Only the DRAM modules available to applications need to be active, and other DRAM modules can be transited into low-power modes. Levy et al. [51] propose a compiler-directed optimization that performs analysis and energy-aware data allocation on dynamic variables instead of static global variables.

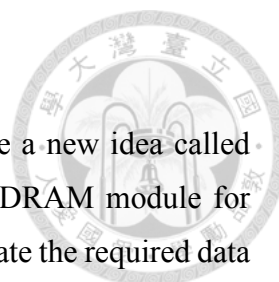
2.1.1.2 Temporal Alignment of Memory Requests

Because the DRAM modules can not be accessed when they are in the non-operational low-power modes and there are power overheads when transiting between the active mode and the low-power modes, transiting the DRAM modules into a low-power mode makes sense only when these DRAM modules can be idle and stay in the low-power mode long enough to save energy that compensates the energy overheads of transiting the DRAM modules into the low-power mode and reactivating the DRAM modules [52]. Instead of altering the data placement to change the access patterns of memory requests spatially to prolong the idle periods of a subset of DRAM modules, another approach is to batch the memory requests into a burst, such that the short idle periods can be clustered into a long idle period.

Pisharath et al. [53] introduce a new kind of on-chip memory module buffers, called Energy-Saver Buffers (ESB), that reside in-between the L2 cache and DRAM modules of the main memory to hold the write requests when the corresponding DRAM modules are in low-power modes, and then flush the write requests to the DRAM modules in burst when the modules are activated for serving read requests, to prolong the idle periods of these DRAM modules. Delaluz et al. [27] present a compiler-based optimization strategy for array-dominated applications. This strategy modifies the execution order of loop



iterations to prolong the idle period of DRAM modules. Wang et al. [18, 19] not only exploit variable partitioning and memory request scheduling for power-aware data allocation, but also use the scheduling policy to achieve temporal alignment of memory requests to capture potential energy savings. Ozturk et al. [31] present the bank-aware cache miss clustering mechanism that is a compiler optimization for clustering cache misses and hits, and increasing bank idleness. Pandey et al. [23] temporally align DMA-memory requests from different I/O buses to the same DRAM module by delaying DMA-memory requests directed to a low-powered DRAM module and gathering enough DMA-memory requests that can fully utilize the active cycles of the DRAM module before activating this DRAM module. Hur et al. [28] propose Power-Aware Adaptive History-Based (PA-AHB) scheduler that groups the memory commands sent to the same DRAM module in the memory queue as close as possible to cluster the memory commands, and they also introduce an adaptive memory throttling mechanism that blocks memory commands inside the memory controller for some fixed periods to allow DRAM modules to remain in low-power modes for long periods of time. Amin et al. [26] introduce a cache replacement policy, called Rank-Aware REplacement (RARE), that tries to avoid replacing the data blocks which map to the prioritized DRAM modules, so that prioritized DRAM modules receive less traffic and could be transitioned into deeper low-power modes for longer periods of time. They also propose Rank-Aware Write Buffer (RAWB) that buffers the write requests going to a DRAM module and then issues these requests as a batch to prolong the idle time. Kant [29] proposes to batch the memory requests by periodically making the DRAM modules inactive directly, so that newly arriving requests queue up and the DRAM modules can be put into low-power modes to improve energy efficiency. Kim et al. [30] implement a DRAM power-aware rank scheduling policy that changes the behaviors of the last-level cache and the memory controller. The last-level cache then prefers replacing dirty blocks that map to active ranks, and avoid replacing dirty blocks that map to ranks in deep low-power modes, so that the overheads of bringing the corresponding ranks back to the active mode for serving the write requests can be minimized. The memory controller tries to buffer write requests that map to ranks in low-power modes, and issues these requests in a batch when the corresponding rank is transitioned to another power mode to minimize the overheads of state transitions.



2.1.1.3 *Data Re-computation*

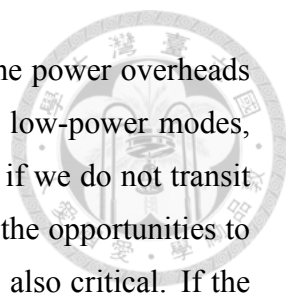
Besides the aforementioned works, Koc et al. [54] introduce a new idea called data re-computation that tries to avoid reactivating a low-powered DRAM module for accessing the required data by performing extra computation to calculate the required data according to the data in already active DRAM modules to improve energy efficiency.

2.1.1.4 *Thermal Drawback of Request Clustering*

The aforementioned methods try to cluster the memory requests issued to DRAM modules in spatial or in temporal, so that the memory requests are served by a subset of DRAM modules in a period of time in burst to prolong the idle periods of DRAM modules. However, during the burst of memory requests, the DRAM modules that serve these requests have high power density due to high dynamic power consumption, and raise their own operating temperature. This may cause thermal emergency on active DRAM modules and dynamic thermal management would be required to prevent overheating. In this case, the thermal issue and the following invocations of dynamic thermal management may further degrade the performance of the memory system. But, these methods do not take the adverse thermal effect into consideration. Therefore, these methods are not suitable to modern high performance and high throughput memory systems that already approach their thermal limitation without these memory request clustering policies, because the performance degradation due to dynamic thermal management may hurt the system performance too much. In contrast, the PPT framework in this dissertation not only focuses on the tradeoff between performance and power, but also takes the thermal effect into consideration to avoid DRAM overheating and performance degradation from invocations of aggressive dynamic thermal management.

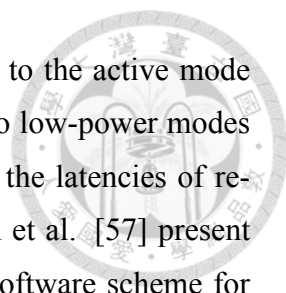
2.1.2 **Power Mode Decision**

In addition to how to prolong the idle periods of the DRAM modules, another important issue of utilizing DRAM low-power modes to save energy is when we should transit the idle DRAM modules into which low-power mode, and when we should reactivate the low-powered DRAM modules back to the active mode. The timing to transit idle DRAM modules into low-power modes is critical. If we transit the idle DRAM mod-

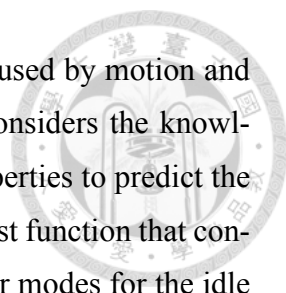


ules into low-power modes when there are only short idle periods, the power overheads of power state transitions may compensate the energy savings from low-power modes, and the low-power mechanisms fail to save energy. On the contrary, if we do not transit the idle DRAM modules into low-power modes, then we may waste the opportunities to save energy. The timing to activate low-powered DRAM modules is also critical. If the DRAM modules are not activated when the memory requests arrive, the memory requests queue up and the system performance degrades due to long memory latency. On the contrary, if the DRAM modules are activated when they are still idle, the memory system just spends unnecessary energy consumption. Therefore, the decision to perform power state transitions for DRAM modules is quite important for the energy efficiency. Another important decision is which low-power mode we should transit the idle DRAM modules into. A deeper low-power mode that consumes less power usually has more power and latency overheads of state transitions, and needs to stay in the low-power mode longer to compensate the overheads. So transiting the idle DRAM modules that have long idle periods into a shallow low-power mode does not maximize the energy savings, and transiting the idle DRAM modules that have short idle periods into a deep low-power mode may cause adverse effect on performance and power due to its high overheads. Therefore, we should select the target low-power mode carefully. The followings are some works that study these issues.

Delaluz et al. [9, 11, 12] propose several compilation techniques and hardware-assisted approaches to perform power state transitions according to program behavior analysis and prediction of inter-access times of DRAM modules. Fan et al. [55] investigate several memory controller policies that manipulate DRAM power states for cache-based systems by developing an analytic model that approximates the idle time of DRAM modules. They find that the simple policy which immediately transits the DRAM modules into low-power modes when they become idle is superior to more sophisticated policies that make decision according to the prediction of the idle times of DRAM modules. Delaluz et al. [56] modify the scheduler in the operating system to direct the power mode transitions by keeping track of the accesses that map to different DRAM banks in a Bank Usage Table for each process in the system. When a process is scheduled, the DRAM banks that do not be accessed by the process are transited into low-power modes. Huang et al. [17] propose a power-aware virtual memory that promotes the DRAM modules



which have some pages mapped into the address space of a process to the active mode when the process is executing, and demotes other DRAM modules to low-power modes to save energy. This approach uses the context switch time to hide the latencies of re-synchronization for minimizing performance degradation. Pisharath et al. [57] present a hardware-based dynamic threshold scheme and a query-directed software scheme for database systems. The hardware-based scheme decides the power states of DRAM modules according to the length of their idle periods, and makes the system adapt to the access patterns of queries. The software-based scheme utilizes the information about the query access patterns from query optimizer that generates the execution plan of queries to transit DRAM modules to desired power states proactively. Lyuh et al. [58, 59] propose a compiler-directed integrated approach to maximally utilize the low-power modes of multiple DRAM modules by solving assignment of variables to DRAM modules, scheduling of DRAM access operations and determinations of DRAM module power modes. Huang et al. [60, 61] present a cooperative software-hardware power management that exploits hardware-controlled power management approach to implement fine-grained and highly-adaptive control mechanism, and uses OS-controlled approach to track system and process state information to predict the memory access patterns that constantly change due to interleaved execution of different processes. Li et al. [62] improve the commonly used control algorithms that dynamically transit DRAM modules into low-power modes after the modules are idle for a certain threshold period of time, by adjusting the thresholds periodically according to available slack and recent workload characteristics. The proposed algorithm has a performance guarantee and is better than previous hand-tuned algorithms. Zheng et al. [63] investigate the combinations of row-buffer management policies and memory low-power modes for different workloads. For memory-intensive workloads, the open-page policy is preferable, but the close-page policy using slow precharge power-down mode provides a better tradeoff in terms of performance and power efficiency than the open-page policy for memory-moderate and compute-intensive workloads. Bi et al. [64] manipulate the power states of the DRAM modules that are dedicated to the buffer cache for file I/O. The DRAM modules are transited into low-power modes when the I/O operations are done, and re-activated when they are predicted to be accessed by the on-going file I/O system calls before the actual data in the buffer cache are read from the DRAM modules to hide the delay of re-synchronization. Zatt et al. [65] introduce a

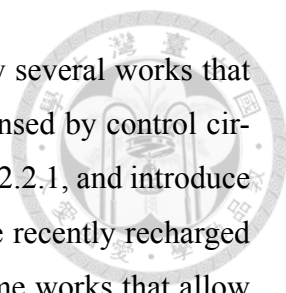


low-power architecture for the on-chip multi-banked video memory used by motion and disparity estimation in multiview video coding. The architecture considers the knowledge of motion and disparity estimation algorithm and the video properties to predict the memory access patterns of each macroblock. Then, they propose a cost function that considers the wake-up overheads to determine the appropriate low-power modes for the idle DRAM modules. Shafique et al. [66] improve Zatt's work by considering texture, motion and disparity properties of objects and their correlations in the 3D-neighborhood, and propose a scheme that groups different macroblocks and predicts the highly-probable motion/disparity search direction to decide which DRAM module should be transited into low-power modes. Mukundan et al. [67] design a self-optimizing scheduling policy that can decide how to schedule DRAM actions including transiting DRAM modules into low-power modes and re-activating DRAM modules to achieve a desirable tradeoff between performance and power efficiency. Chandrasekar et al. [68] propose memory power-down strategies for real-time systems to reduce memory energy consumption and guarantee real-time memory performance at the same time. They also present an algorithm that can select the most energy-efficient low-power mode dynamically.

The PPT framework only manipulates the access patterns of the workloads to prolong the idle periods of DRAM modules and balance power density across DRAM modules to avoid overheating, so it basically can cooperate with every policy that decides the power modes of DRAM modules. In this dissertation, the PPT framework utilizes the open-page row-buffer management policy for DDRx-SDRAM and adopts the simple policy that transits the idle DRAM modules into the low-power mode as deep as possible when the DRAM modules are idle immediately. However, the PPT framework is also able to adopt a more sophisticated policy that provides a better tradeoff between performance and power, but this is out of the scope of this dissertation.

2.2 Refresh Power Reduction

Because the charges in the capacitors in DRAMs leak away gradually and the data in DRAMs may loss due to the leakage, refresh operations are required to recharge the capacitors to guarantee the data correctness. Therefore, the mechanisms that reduce the refresh operations all need to meet the same requirement that the results of program ex-

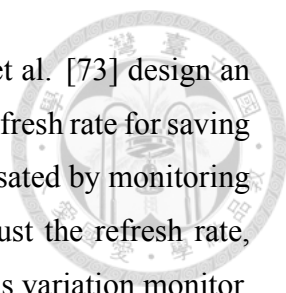


ecutions can not be significantly changed due to the leakage. I show several works that propose to refresh DRAM cells according to the leakage currents sensed by control circuits instead of utilizing a fixed worst case refresh interval in Section 2.2.1, and introduce studies that skip the refresh operations to the DRAM cells which are recently recharged by memory accesses in Section 2.2.2. In Section 2.2.3, I discuss some works that allow charges of DRAM cells to leak away to reduce refresh operations when the data in these DRAM cells have some special properties that make the data loss insignificant or recoverable. Papers presented in Section 2.2.4 apply different refresh intervals to DRAM regions according to their own worst case refresh intervals instead of setting the refresh interval according to the worst case of the whole DRAM system. I introduce the works that reduce refresh operations by prolonging the refresh interval and correcting retention errors by ECCs to guarantee data correctness in Section 2.2.5. Then, the studies which adjust the refresh interval dynamically to adapt to system perturbations that change the leakage rates of DRAM cells are discussed in Section 2.2.6.

2.2.1 Refresh Reduction by Sensing Leakage Current

Since the refresh operations of DRAMs are designed to recharge the DRAM cells for preventing data loss due to leakage of DRAM cells, the most straightforward approach to reduce refresh operations is to refresh the DRAM cells right before the retention errors occur.

Nyathi et al. [69] present a CMOS circuitry that senses the integrity of stored data by using differential amplifiers that provide the difference between a degrading stored voltage and a reference voltage. The difference is converted to an output that is used as the refresh trigger. Cho et al. [70] monitor the leakage of memory cell data and perform refresh operations according to the results of monitoring when the DRAM modules are in the self-refresh mode. Hsu et al. [71] use a monitor cell that represents the average cell or worst cell leakage condition in the real array, and adjust the refresh interval according to the leakage condition of this monitor cell to minimize the refresh power. Burgan et al. [72] control the refresh rate by utilizing a set of test cells. One approach is to use test cells with the same capacity but refresh them at different rates, and the other approach is to use test cells with different capacities but refresh them at the same rate. Then, the



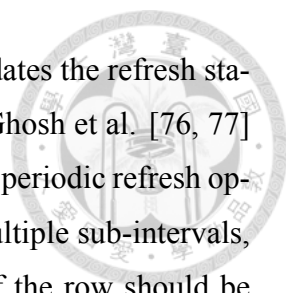
refresh rate is adjusted according to the states of the test cells. Tsai et al. [73] design an adaptive refreshing circuitry for DRAMs to automatically adjust the refresh rate for saving standby power. In this design, the temperature variations are compensated by monitoring the voltage drop of memory cells with a voltage comparator to adjust the refresh rate, and the process drifting problem is compensated by utilizing a process variation monitor. Tran et al. [74] connect many dummy cells on boundaries together to form a leakage current sensor that emulates the leakage currents in memory cells to provide information for adjusting the supply voltages and the refresh rate.

All aforementioned techniques change the design of conventional DRAMs such as DDRx-SDRAM. Since the manufacturing procedure of the modern conventional DDRx-SDRAM is highly optimized, every change to the DDRx-SDRAM may increase a lot of cost of DRAMs. Therefore, these techniques are not suitable for current DRAM systems due to their cost to change the DRAM architecture. On the contrary, the SECRET framework does not change the DRAM architecture and only alters the operating system and the memory controller with negligible overheads. So, the SECRET framework is more preferable than these techniques that reduce refresh operations by sensing leakage currents.

2.2.2 Refresh Reduction by Considering Access Pattern

DRAM cells store data by keeping charges in their capacitors, and the data can be read by sensing the charges. However, when the sense amplifiers sense the charges, the charges go away and there is no valid data in the DRAM cells after the read operations. Therefore, the valid data need to be written back to the DRAM cells after read operations. Since a refresh operation is only the combination of reading the data from the DRAM cells and then writing the data back to the DRAM cells, every read or write request to the DRAM cells can recharge the DRAM cells as the refresh operation. Therefore, the DRAM cells that are recently read or written do not need to be refreshed because they are just recharged. Based on this observation, there are several works which skip the refresh operations for the DRAM cells that are recently recharged to reduce refresh operations.

Emma et al. [75] propose to only refresh the DRAM cells that are not read from or written to within an allotted data retention time for reducing refresh operations of a

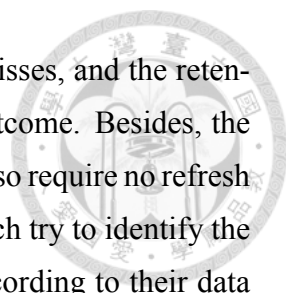


DRAM cache by using a restore tracking system that records and updates the refresh status of the data entries, and invalidates data entries that are expired. Ghosh et al. [76, 77] present the Smart Refresh mechanism that eliminates the unnecessary periodic refresh operations. This mechanism divides the normal refresh interval into multiple sub-intervals, and employs a time-out counter for each memory row to indicate if the row should be refreshed in the next sub-interval. The counter is reset when the row is read or written, so periodic refresh operations that follow the read or write requests are eliminated to reduce power consumption. Emma et al. [78] reduce the refresh operations of DRAM caches by utilizing a time stamp in each directory entry to indicate if the corresponding line is read or written in its current retention window. When a line is read or written in the retention window, the refresh engine would skip the next refresh operation to the line. Agrawal et al. [79] focus on on-chip eDRAM caches and propose Polyphase that is quite similar to Smart Refresh. Polyphase partitions the retention time into a fixed number of equal intervals called Phases and records the last access phase of each cache line. With Polyphase, a line is refreshed only when the same phase arrives in the next retention period.

Basically, all these mechanisms use the same idea to reduce refresh operations, but different implementations. This idea is quite effective for reducing the refresh operations of DRAM cells that store hot data, because the DRAM cells are read or written frequently and a lot of refresh operations can be eliminated. However, on the contrary, the DRAM cells that store cold but valid data should be refreshed as frequent as normal with these mechanisms, because there are few read or write requests to these DRAM cells. So, these mechanisms only bring limited refresh reduction and power savings when the memory system has large capacity and contains a lot of cold data. However, the SECRET framework can achieve the same refresh reduction for both DRAM systems containing hot data and DRAM systems containing cold data.

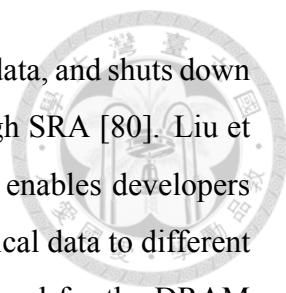
2.2.3 Refresh Reduction by Considering Data Property

The main objective of refresh operations is to recharge the DRAM cells periodically to prevent data loss due to the leakage currents of DRAM cells. Therefore, the DRAM cells that do not store valid data actually require no refresh operations. The refresh operations to the cached clean data or non-critical data can also be eliminated since



the accesses to incorrect cached clean data can be treated as cache misses, and the retention errors in non-critical data only incur little impact to the final outcome. Besides, the DRAMs storing values which do not change due to leakage currents also require no refresh operations. Based on these observations, there are several works which try to identify the memory segments that can tolerate the effect of leakage currents according to their data properties, and disable the refresh operations for these memory segments to reduce refresh operations of the DRAM systems.

Ohsawa et al. [80] propose Selective Refresh Architecture (SRA) that allows DRAM rows to be refreshed selectively. When the data in a DRAM row is out of the lifetime, the data is no more required. In this case, the corresponding refresh flag of the row in the memory controller is reset and the row will not be refreshed any more. In SRA, the refresh flag can be set or reset by the operating system, the memory management unit or the compiler according to the data lifetime analysis. Vargas [5] presents the partial-array self-refresh (PASR) mechanism that reduces the number of refresh operations by refreshing only a part of the memory array when the part contains valid data and other parts do not contain any data. Venkatesan et al. [81] propose Retention-Aware Placement in DRAM (RAPID) that reduces refresh operations by concentrating data into physical pages with long retention times and applying a long refresh interval according to the shortest retention time of used physical pages. Moshnyaga et al. [82] present an OS-controlled policy that stops refreshing for the DRAM banks allocated for the DRAM-based swap-caches if the data in the caches are clean and not accessed for a long time. This approach reduces refresh operations at the expense of retrieving the required data from the next level of memory hierarchy when the data in the swap-caches vanish due to stopping refreshing. Emma et al. [78] reduce the refresh operations of the DRAM write-through caches by stopping refreshing and protecting the data by using Berger code. This scheme lets the useful data be refreshed by the innate reference pattern and lets the stale data decay. If the decayed data are read, the Berger code would show errors and the read request for the decayed data is treated as a cache miss. Isen et al. [83] stop refreshing the DRAM regions that are not allocated or only store invalid data to reduce refresh operations, because the data in these regions are inconsequential to the correct execution of the program. Kim et al. [84] present an operating system-directed scheme to reduce DRAM refresh operations. This scheme analyzes the page structures that maintain the status of physical pages



in the operating system to indicate whether a DRAM row keeps valid data, and shuts down the refresh operations for the rows that do not store valid data through SRA [80]. Liu et al. [85] introduce an application-level technique called Flicker that enables developers to specify critical and non-critical data, allocates critical and non-critical data to different physical DRAM modules separately, and applies a long refresh interval for the DRAM modules containing non-critical data to reduce the refresh operations at the expense of a modest data corruption in the non-critical data with little or no impact on the final outcome of the application. Agrawal et al. [79] propose a refresh reduction scheme for eDRAM caches to not only eliminate the refresh operations for invalid cache lines, but also stop refreshing the cold valid cache lines that are not accessed for a while.

There is a special mechanism that is designed based on the observation that the leakage currents of DRAM cells are always unidirectional. So, the voltages of capacitors of DRAM cells always change from high to low due to the leakage currents. Assuming the high voltage represents the one value, and the low voltage represents the zero value, the capacitors storing zero values do not need to be refreshed because their values would not change due to leakage. According to this observation, Patel et al. [86] propose a low-overhead refresh reduction mechanism based on selectively skipping the refresh operations for the DRAM cells that store zero bits. They add a limited amount of redundant storage and logic to indicate the DRAM regions that only contain zero values, and eliminate refresh operations to these regions to reduce refresh power consumption. This mechanism can cooperate with other refresh reduction mechanisms if there are some DRAM regions containing only zero values.

These mechanisms reduce refresh operations on the DRAMs containing invalid data, all zero values, non-critical data, or clean data that are duplicated somewhere else. Their effectiveness highly depends on the properties of the data in DRAMs. If the DRAMs are fully utilized, all data in these DRAMs are critical, no DRAM regions contain all zero values and no retention errors are allowed, these mechanisms can not reduce any refresh operations. However, the SECRET framework can reduce the refresh operations without data decay, so it works for fully utilized DRAMs storing critical data as well.

2.2.4 Multi-Period Refresh

There are variations in the retention times of DRAM cells, and the refresh interval of a DRAM system is usually bound by the shortest retention time of DRAM cells to avoid data loss. However, most DRAM cells in a DRAM system have long retention times and only a few DRAM cells have very short retention times. For the DRAM cells that have long retention times, frequent refresh operations based on the worst-case refresh interval are over-provisioned and unnecessary. So, a promising approach to avoid unnecessary frequent refresh operations for DRAM cells that have long retention times is to partition the DRAM system into multiple regions and apply a refresh interval to each region according to the shortest retention time of DRAM cells in that region instead of in the whole system.

Ohsawa et al. [80] propose Variable Refresh Period Architecture (VRA), that allows multiple refresh periods in the DRAM system. Architects can apply the most appropriate refresh period to each row to reduce refresh count. Takase et al. [87] present the Additional Refresh scheme that applies the refresh interval eight times longer than the normal refresh interval to reduce refresh operations, and adds additional refresh operations selectively to the rows containing weak retention DRAM cells to avoid retention errors. Kim et al. [88] also utilize a similar approach that uses a collection of discrete refresh periods to selectively refresh DRAM regions. DRAM regions comprising DRAM cells with short retention times are still refreshed with the minimum refresh interval, but other regions can be refreshed with long refresh intervals. They also develop an algorithm to maximize the refresh power reduction by selecting appropriate refresh periods according to the number of refresh regions, the number of refresh periods and the number of cells in the refresh regions. Kim et al. [89, 90] improve previous multiple refresh period schemes by applying the most appropriate refresh intervals to small refresh blocks instead of DRAM rows to increase the worst case data retention times, extending the retention time of each refresh block by adding a swap cell, and developing a polynomial-time algorithm to compute an optimal set of refresh intervals for the block-based multi-period refresh. Liu et al. [7] propose RAIDR (Retention-Aware Intelligent DRAM Refresh), that is a low-overhead mechanism to identify and skip unnecessary refresh operations according to the knowledge of cell retention times. RAIDR utilizes Bloom filters to minimize the overhead of recording the rows having weak retention cells and efficiently group DRAM

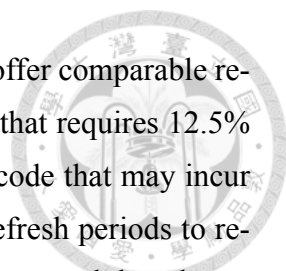
rows into retention time bins, and applies a different refresh rate to each retention time bin. So, the rows having weak retention cells are refreshed as frequent as normal, and other rows are refreshed less frequently. RAIDR requires no modification to DRAMs and a little modification to the memory controller.

These multiple refresh period mechanisms can cooperate with the SECRET framework. After applying different refresh rates to DRAM regions, the refresh interval of a DRAM region is still bound by the shortest retention time of DRAM cells in the region. So, further prolonging the refresh interval of the region would incur data loss due to retention errors on the weak retention cells. In this case, utilizing the SECRET framework can allow the region to use a long refresh interval to further reduce refresh power and avoid data loss by correcting the retention errors on the weak retention cells in the region.

2.2.5 ECC for Refresh Reduction

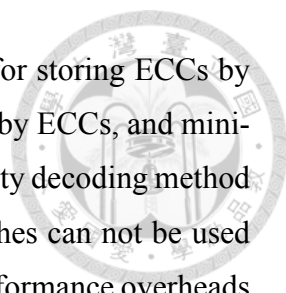
The main drawbacks of prolonging the refresh interval to reduce refresh power are retention errors on the weak retention cells that are not refreshed in time. So, this problem can be overcome by using ECC that corrects retention errors, and prolonging the refresh interval can save power without data loss in this case. This approach is one kind of Better Than Worst-Case designs to system implementation [91]. Better Than Worst-Case designs relax design constraints to reduce the physical design challenges and create opportunities to improve performance or energy efficiency. In refresh power reduction, the ECC approaches relax the constraints to allow retention errors in the DRAM systems, and create the opportunities to prolong the refresh interval for refresh power savings.

Katayama et al. [92] apply a powerful one-shot Reed-Solomon ECC to protect the data stored in DRAMs, so that they can prolong the refresh interval to greatly reduce the refresh power while still maintaining data integrity. The Reed-Solomon code provides up to double-symbol error correction capability and reasonable triple-symbol error detection capability. However, decoding Reed-Solomon code is quite slow and would incur noticeable performance degradation. The SECRET framework is more preferable than this approach, because the SECRET framework only suffers negligible performance degradation. Katayama et al. [93] then analyze various configurations of error correction codes for DRAM data retention. They find that combining long and short error correction



codes can reduce the parity area to 1% of the total memory size, and offer comparable reliability and adaptability as the aforementioned Reed-Solomon code that requires 12.5% parity area. However, the hybrid approach still uses Reed-Solomon code that may incur performance degradation. Kim et al. [88] not only utilize multiple refresh periods to reduce refresh power, but also use error correction codes to recovery corrupted data due to retention errors to further extend the refresh interval. Klein et al. [94] propose a reduced power refresh mode. The ECC syndromes are generated and stored before transiting the DRAM modules into the reduced power refresh mode. In the reduced power refresh mode, DRAM cells are refreshed at a relatively slow rate. When transiting DRAMs from the reduced power refresh mode to the active mode, stored data and syndromes are both read and retention errors are corrected according to the syndromes. Then, the corrected data are written back. Kim et al. [95] design a 512Mb mobile SDRAM with on-chip ECC that increases the self refresh period at standby state by about 6 times and reduces the self refresh current to be less than $100\mu\text{A}$ at 85°C . Emma et al. [78] propose to use SECDED (single error correction, double error detection) ECC for eDRAM caches to reduce the refresh power. When reading cache lines, the lines with only one error can be corrected directly by the ECCs, and the requests to the lines with two or more errors are treated as cache misses. They also derive a statistical expression to calculate the desired refresh interval with ECC. Cha et al. [96] reduce the size of check bits by using a codeword which has a length that is not a power of two. Compared to the use of 2^n data bits in 1Gb DRAMs, the proposed codeword can respectively achieve 3.4% and 4.7% reduction in check bit and register overheads for self-refresh schemes with ECCs. Wilkerson et al. [97] use BCH code to provide multi-bit correction for eDRAM caches to reduce refresh power. They reduce performance overheads by using a low-complexity decoding method for cache lines with no more than one error, and minimize the area overheads for storing ECCs by increasing the data size protected by BCH. However, this approach requires large bandwidth to retrieve the whole data for error detection and correction, and the optimization that reduces the required bandwidth is only suitable for on-chip caches, not for off-chip memory systems.

Most of these mechanisms utilize ECC to protect all data in the DRAM systems from retention errors for refresh power reduction. So, the area overheads for storing the ECCs are quite large, and decoding all read data incurs noticeable performance degra-

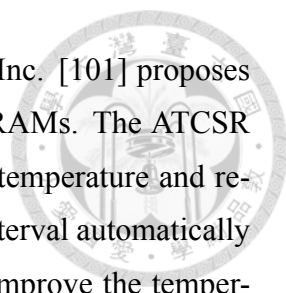


dation. Although some of them [93, 97] reduce the area overheads for storing ECCs by combining long and short ECCs or increasing the data size protected by ECCs, and minimize the performance overheads of decoding by using a low-complexity decoding method for most cases where there is no more than one error, these approaches can not be used on conventional off-chip DRAM systems with negligible area and performance overheads due to the bandwidth limitation. However, for the target off-chip DRAM systems, the SECRET framework can achieve comparable refresh power reduction as these approaches with negligible area and performance overheads by selectively protecting the data blocks that may have retention errors only.

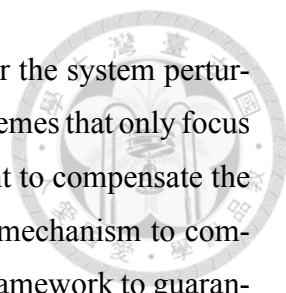
2.2.6 Adaptive Refresh Interval

The leakage currents of DRAM cells may change due to various system perturbations, such as temperature variations. So, the refresh interval should also be adjusted to adapt to the system perturbations. For example, the leakage currents of DRAM cells increase as the operating temperature of DRAMs rises, and the data retention times of DRAM cells decrease as well. Therefore, the worst case refresh interval is limited by the shortest data retention time at the highest operating temperature in the DRAM specifications. However, in most cases, the operating temperature of DRAMs is much lower than the highest operating temperature in the worst case, and it is possible to utilize a long refresh interval by detecting the operating temperature of DRAMs. There are several works that focus on how to adjust the refresh interval against temperature variations.

Murotani et al. [98] use a memory circuit that comprises an oscillator circuit having a frequency characteristic with a positive temperature coefficient. The oscillator circuit generates refresh signals based on its oscillation period to achieve a temperature-compensated refresh function. Kagenishi et al. [99] introduce a self-refresh controller with a temperature detecting circuit composed by a CMOS type differential amplifier and voltage dividers of resistors that generate the input signals to the amplifier. The resistances of resistors vary with the operating temperature, and the output signals of the detecting circuit are used to select a suitable refresh interval corresponding to the operating temperature. Ruckerbauer et al. [100] use temperature sensors for DRAM arrays to indicate the operating temperature of the DRAM arrays, and adjust the refresh rate according



to the operating temperature of the DRAM arrays. Elpida Memroy, Inc. [101] proposes Auto Temperature Compensated Self Refresh (ATCSR) for Mobile RAMs. The ATCSR function utilizes a built-in temperature sensor to detect the ambient temperature and reduces DRAM refresh power consumption by adjusting self-refresh interval automatically according to the ambient temperature variations. Kim et al. [102] improve the temperature sensor for controlling the self-refresh interval of mobile DRAMs by presenting a low-cost CMOS temperature sensor that is highly area-efficient, simple and easy for IC implementation as compared to traditional temperature sensors based on bandgap reference. Besides adjusting the refresh interval according to the operating temperature, they also adjust the refresh interval by detecting the retention errors in DRAM arrays according to ECCs. When retention errors occur, it means that the refresh interval is not short enough to maintain the correctness of data stored in DRAM cells. In this case, the refresh interval should be reduced. On the other hand, when there are no retention errors, it is possible to prolong the refresh interval for power reduction. Katayama et al. [92] find that system perturbations affect the number of retention errors in DRAM systems. So, they propose to adjust the refresh interval according to the number of retention errors by attempting to keep the number of retention errors constant. They identify some indicator bits that have data retention times a little longer or shorter than the default refresh interval in the profiling phase. When the retention errors in these indicator bits are more than the expected retention errors, the refresh interval is decreased to reduce retention errors. When the retention errors in these indicator bits are less than the expected retention errors, the refresh interval is increased to achieve refresh power reduction. Emma et al. [78, 103] propose a regressive process to find the appropriate refresh interval for each row in DRAM caches by utilizing ECCs. In this process, the refresh interval for each row is set to a nominal value such as the worst case initially. The refresh interval of each row is regressed by one time step after every program execution interval if there are no retention errors found in that row. When any errors are found in a row, the retention errors are corrected by ECCs, the refresh interval of the row is set to the previous one that incurs no retention error, and the regressive process for this row stops. This regressive process can be performed periodically to adjust the refresh interval to adapt to the system perturbations. Since each row can have its own refresh interval after the regressive process, this method is also a multiple period refresh mechanism.

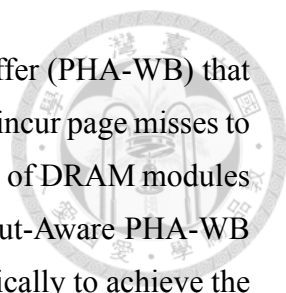


These refresh interval adjustment mechanisms which consider the system perturbations can cooperate with other aforementioned refresh reduction schemes that only focus on reducing the refresh operations under a given runtime environment to compensate the effect of system perturbations. Since the refresh interval adaptation mechanism to compensate the temperature variations is also required for the SECRET framework to guarantee the correctness, I integrate a refresh interval adjustment mechanism similar to the one proposed by Katayama et al. [92] into the SECRET framework.

2.3 Dynamic Thermal Management for DRAMs

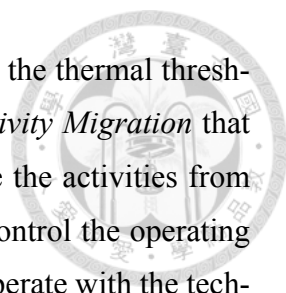
In the past, the dynamic thermal management policies often focus on the processors that usually have the highest power density in the computer systems. However, in the recent years, because the speeds and capacities of DRAM modules increase significantly, the power density of DRAM modules also rises. So, the operating temperature of DRAM modules is approaching the thermal constraints of DRAM cells. To protect the memory system from thermal emergency, several dynamic thermal management techniques for the memory subsystems have been proposed.

Iyer et al. [42] find that the memory operating temperature is starting to exceed the cooling capabilities of DRAM modules in mobile systems. So, they propose to monitor the operating temperature of DRAM modules and perform memory access throttling if the DRAM modules overheat. They implement memory throttling techniques in platforms built on Intel Centrino Duo mobile technology to maximize performance while keeping the memory subsystem within its thermal limits. Lin et al. [44] propose adaptive core gating and coordinated DVFS that throttle the processors which are the source of memory activities to reduce the memory requests and control the operating temperature of DRAM modules. When the DRAM modules are overheated, adaptive core gating activates clock gating on selected processor cores, and coordinated DVFS scales down the frequency and voltage levels of processor cores. These approaches result in smoother program execution, higher system performance and better power efficiency than throttling the memory system directly. Lin et al. [45] then implement the dynamic thermal management policies on real server systems to evaluate their effectiveness. They find that the operating temperature of DRAM modules is also affected by the heat dissipation of CPU, and this significant



factor was ignored. Liu et al. [46] propose Page Hit Aware Write Buffer (PHA-WB) that improves DRAM page hit rate by buffering write requests which may incur page misses to reduce the power consumption and alleviate the operating temperature of DRAM modules without performance penalty. Liu et al. [47] then develop Throughput-Aware PHA-WB (TAP) that configures the write buffer for different workloads dynamically to achieve the best tradeoff between the DRAM power savings and the power overheads of the write buffer. Ayoub et al. [48] develop a proactive thermal management policy that cooperates with commonly used power-aware page allocation which allocates pages to a subset of DRAM modules and transits other DRAM modules into low-power modes. When the operating temperature of the DRAM modules that contain the pages of working sets is predicted to approach the thermal limitation, the proposed policy migrates the pages to colder DRAM modules and transits the hotter ones into the self-refresh low-power mode. Liu et al. [49] find that the operating temperatures of DRAM chips in the same DIMM can vary by over 10°C according to their physical positions. So, they try to minimize this variation to reduce the peak temperature of DRAM chips by alternating the DIMM organization to enable independent DRAM traffic to hot and cool DRAM chips on the same DIMM. They also develop a cache line replacement policy that evicts lines to cool DRAM chips first, a memory write buffer that improves the access efficiency of the overheated DRAM chips, and a page allocation policy that allocates pages to cool DRAM chips first to further reduce peak temperature of the memory system. Meng et al. [104] propose an optimization technique to maximize throughput while maintaining the power and temperature constraints for 3D multicore systems with on-chip DRAMs by monitoring workload behaviors through performance counters and adjusting the voltage-frequency settings dynamically to adapt to varying program phases. Meng et al. [105] then develop a thermal-aware page allocation policy for 3D systems with stacked DRAM banks by allocating frequently accessed pages to DRAM banks with low temperature and rarely accessed pages to DRAM banks with high temperature according to static analysis or dynamic adjustment.

The PPT framework uses *Group Switching* that is a quite different way to control the operating temperature of the memory system by changing the active DRAM modules periodically to reduce power consumption and balance the temperature of DRAM modules. This approach does not need additional hardware resources such as an extra write buffer, and only incurs negligible performance penalties with no throttling for CPUs or

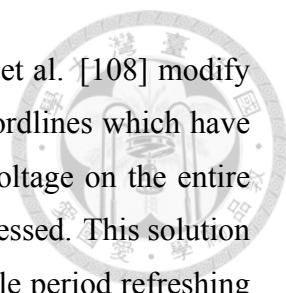


the memory system. However, the DRAM modules may still exceed the thermal threshold when they are active. In this case, the PPT framework uses *Activity Migration* that is similar to the mechanism proposed by Ayoub et al. [48] to move the activities from the overheated DRAM modules to other cool modules. To further control the operating temperature of the memory system, the PPT framework can also cooperate with the techniques proposed by Liu et al. [49], and Meng et al. [104, 105]. But that is out of the scope of this dissertation.

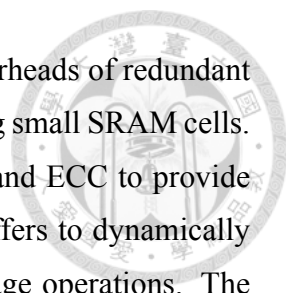
2.4 Reliable Low-Voltage Operation for SRAM Cache

Besides the aforementioned approaches that reduce refresh power of DRAMs in Section 2.2, one kind of related mechanisms that solve a problem similar to the refresh reduction is to reduce the power consumption of processors by reducing the supply voltage. Due to manufacturing-induced parameter variations, reducing supply voltages of processors may cause memory circuits to fail. Therefore, to avoid the failures in memories, supply voltage scaling for power reduction is limited by a minimum voltage, called V_{ccmin} . The V_{ccmin} is a lower bound supply voltage for memory circuits in processors similar to the upper bound of the refresh interval for DRAMs. The V_{ccmin} is usually bound by large memory structures, such as caches in processors. If the supply voltage is lower than the V_{ccmin} , there will be memory cell failures in caches. That is similar to prolonging the refresh interval beyond the shortest retention time of DRAM cells with the drawback of retention errors. A low V_{cc} for the processors or a long refresh interval for the DRAMs both save energy at the expense of some errors in the memories. So, reducing the V_{cc} and prolonging the refresh interval have quite similar tradeoff between power savings and error handling. Now, I introduce some techniques that provide reliable low-voltage operations to processors for power reduction.

Wilkerson et al. [106, 107] propose two techniques to handle the faulty bits in the caches under low supply voltages. The Word-disable scheme combines two consecutive cache lines to form a single line where only error-free words are utilized, and the Bit-fix scheme uses a quarter of the ways of each set to store locations and correction information for faulty bits in other ways of the set. The Word-disable scheme and the Bit-fix scheme require 50% and 25% area overhead, respectively. These schemes are not good enough



for refresh power reduction due to very high area overheads. Sasan et al. [108] modify the peripheral circuitry of the SRAM to selectively overdrive the wordlines which have weak cells to allow reducing the power by decreasing the supply voltage on the entire array but maintain the correctness when rows with weak cells are accessed. This solution for reliable low-voltage operations on caches is similar to the multiple period refreshing for refresh reduction. Abella et al. [109] disable faulty sub-blocks to tolerate high faulty bit rates in caches under the low supply voltage. With this scheme, the fault-free blocks can be used with parity protection or the blocks having up to one error can be used with SECDED ECC protection. They consider four different granularities of disabling faulty blocks as cache section, cache line, sub-block and sub-subblock. This approach still has noticeable area overheads. Ansari et al. [110] reconfigure the internal organization of the cache architecture to tolerate SRAM failures in the ultra low voltage region. The cache lines are divided into multiple data chunks, and a chunk having faulty bits is labeled faulty. The cache lines are partitioned into groups carefully to make sure that every two lines in the same group do not have faulty chunks at the same position. Then, one of the lines in a group is used as the redundant line for the other lines in the same group. Each cache read or write needs to access the target cache line and the corresponding redundant line, and composes a fault-free block by selecting the non-fault chunks appropriately. This approach may cause significant area overheads due to the redundant line in each group. Chishti et al. [111, 112] propose multi-bit segmented ECC (MS-ECC) to tolerate both persistent and non-persistent failures at low voltages. Furthermore, the design of MS-ECC also allows the operating system to adaptively adjust the cache size and ECC capability to adapt to the condition of the system. This idea is similar to using ECC to correct retention errors for refresh reduction. Miller et al. [113] design Parichute that is a powerful error correction technique based on turbo product codes to handle the failures in caches under low supply voltages. Parichute can selectively protect the cache sections that exhibit errors with the strong error correction capability. Parichute is flexible to be disabled to avoid area and performance overheads in high supply voltage operations. However, in near-threshold, there is still only 50% capacity available in the caches with Parichute, and the other capacity is used to store error correction information. Zhou et al. [114] try to minimize the total SRAM area of LLC by orchestrating the size of SRAM cells, the number of redundant cells, and ECC strength while meeting the target yields and V_{dmin} . The redundant cells and ECC can



tolerate the failures that occur in small SRAM cells, and the area overheads of redundant cells and ECC can be compensated by the area reduction from utilizing small SRAM cells. This is an optimization for the approaches that use redundant cells and ECC to provide reliable low-voltage operations. Mahmood et al. [115] use fault buffers to dynamically replace the actively used faulty words in L1 caches under low-voltage operations. The fault buffers are organized as multiple banks to reduce the cost of implementation and can be reconfigured dynamically to adapt to different performance requirement. Choi et al. [116] find that most cache misses in sub-block disable-based methods for reliable low-voltage operations in processors are caused by accessing faulty words under the low supply voltage. So, they try to reduce these cache misses by the proposed word-level sub-block disable-based method that maps the frequently accessed data to non-faulty words. This approach exploits both access and error patterns, and can give the performance of the error-free cache even at a low V_{cc} if these two patterns are always matched in the best scenario. BanaiyanMofrad et al. [117] propose Flexible Fault-Tolerant Cache (FFT-Cache) that utilizes aggressive voltage scaling to reduce power and uses a portion of faulty blocks as redundancy to tolerate other faulty blocks. FFT-Cache tries to sacrifice a minimal number of cache lines to tolerate the maximum amount of defects and avoid performance degradation. Alameldeen et al. [118] use variable-strength error-correcting codes (VS-ECC) to correct the errors in caches under the low supply voltage. In the common case, cache lines with zero or one failure are protected by a simple and fast ECC with little area and latency overheads, and a small number of lines with multi-bit failures use a strong multi-bit ECC with large area and latency overheads. Since only a small number of cache lines require the strong multi-bit ECC, compared to prior multi-bit correcting mechanisms, VS-ECC reduces power without causing large area, latency and bandwidth overheads. However, they still apply ECC to all cache lines. Yang et al. [119] introduce unequal-error-protection error correcting codes (UEPECCs) to improve the reliability of caches under low supply voltages for mobile multimedia applications. Since different bits in the same word are usually not equally significant for mobile multimedia applications, these bits deserve different protection levels. They develop a metric to measure the reliability of a word when different bits are not equally significant and design an optimization algorithm to construct UEPECC that applies different protection levels to these bits according to their significance. UEPECC still protects all cache lines even when some cache lines are totally

error-free.



2.5 Related Work Using Ideas Similar to the PPT Framework

In this section, I will introduce several works that present some techniques similar to the mechanisms in the PPT framework and they cite PPT. Ayoub et al. [48] propose a dynamic thermal management technique that moves frequently accessed pages from hot DIMMs to cold DIMMs when the hot DIMMs approach their thermal threshold, and then transits the hot DIMMs into low-power modes to save energy and alleviate their operating temperature. This mechanism is very similar to the *Activity Migration* mechanism in the thermal control policy of the PPT framework, but performs page migrations proactively according to the thermal prediction. Ayoub et al. [25] then present a memory actuator that reduces the power consumption of the memory system by clustering active memory pages into a subset of DRAM modules, and controls the power dissipation of specific DIMMs by performing page migration. This idea is similar to the power reduction mechanism and the *Activity Migration* mechanism of the PPT framework. But, when some active DIMMs are overheated and no other active DIMMs can accept additional pages, Ayoub's memory actuator provides another alternative that is to spin up the fan to cool the overheated DIMMs in addition to activating a new DIMM. Jia et al. [120] propose the memory affinity aware scheduling (MAS) that is very like the PPT framework but has two improvements. First, MAS considers the shared memory address space between threads, and partitions the threads with shared memory address space into the same group. Second, MAS takes the fairness into account during scheduling. Jantz et al. [121] use a technique similar to *Adaptive Grouping* of the PPT framework to manipulate system performance and power consumption by concentrating or distributing the pages of the executing applications across DRAM modules. But, they perform the hot/cold separation and focus on the hot pages to utilize the active memory space more efficiently. However, this work does not consider the thermal issue of the memory systems.

To the best of my knowledge, the PPT framework is the first work that focuses on the performance, power and thermal issues at the same time for the memory subsystems. Although these aforementioned works have some improvements compared to the PPT framework, their main ideas are still similar to the PPT framework that manages per-

formance, power and temperature of the memory subsystems by threads scheduling and page allocation.





Chapter 3

Introduction to DDRx-SDRAM

In this dissertation, I focus on the memory systems composed of Double-Data Rate 2 (DDR2) or Double-Data Rate 3 (DDR3) SDRAM memories. DDRx doubles the available bandwidth by transferring data at both edges of the clock. DDRx internal transfers from and to the DRAM array read and write twice the number of bits as single data rate SDRAM. DDRx is often packaged as DIMMs, each of which commonly contains 1 or 2 ranks. A memory system can contain multiple channels, and each channel is associated with 1 or 2 DIMMs. A rank is the smallest physical unit for power management.

3.1 DRAM Organization

The DDRx-SDRAM is composed of conventional 2D DRAM arrays, which are rectangular grids of DRAM cells shown in Figure 3.1. By specifying a row address and a column address, the memory controller can read or write a specific DRAM cell inside a DRAM chip. As shown in Figure 3.1(a), each DRAM cell is composed of a transistor and a capacitor connected to the wordline and the bitline. When the wordline is charged, the transistor of the DRAM cell is opened and the cell can be read or written through the bitline. To write the DRAM cell, the capacitor is charged or discharged to high voltage or low voltage through the bitline connected to the write driver. To read the DRAM cell, the charge in the capacitor is passed to the sense amplifier through the bitline. In a 2D DRAM array shown in Figure 3.1(b), once the row decoder selects a row and the wordline is charged, all transistors of the row are activated, and the charges in the capacitors of the DRAM cells are passed to sense amplifiers through the bitlines. Therefore, DRAMs are read-destructive, and all data are only stored in the row buffer when a row is opened.

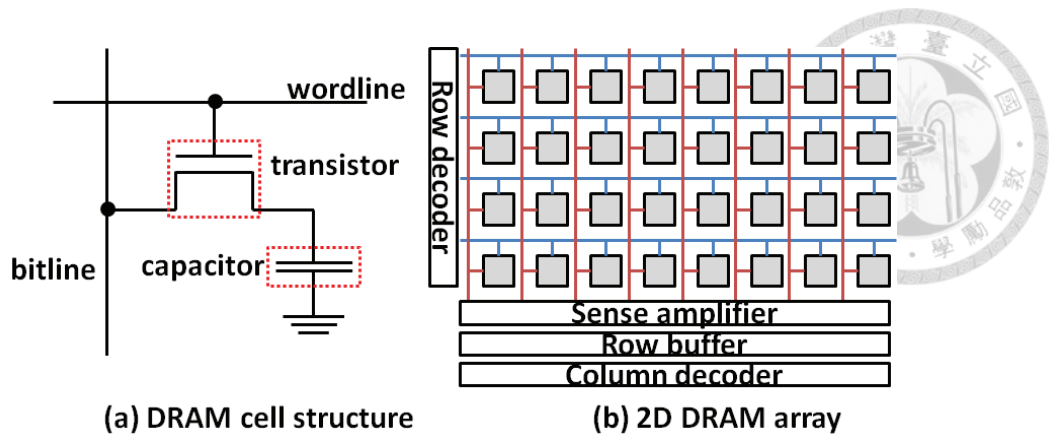


Figure 3.1: The structure of a DRAM cell and a 2D DRAM array.

3.2 Power States of DDR_x-SDRAM

A DDR_x memory device can be in four power states – *active standby*, *precharge standby*, *active power-down* and *precharge power-down* – listed in a decreasing order of power dissipation. Figure 3.2 shows the state transitions among these four power states. A rank can only be accessed when it is in the *active standby* state, where data are stored in row buffers and the clock enable signal is set to HIGH. In this case, all subcomponents of a rank are active. DDR_x has a built-in power control policy. When there is no immediately following request, the memory controller may set the clock enable signal to LOW and a rank transits from *active standby* to *active power-down*. In the *active power-down* state, the row/column decoders and bus drivers are turned off for power reduction. A rank is transferred to the *precharge standby* state after precharge or refresh operations occur. In this case, there is no data stored in row buffers, and row buffers are disabled to save energy. When a rank is in the *precharge standby* state without immediately following request, the rank can transit to the *precharge power-down* state by setting clock enable signal to LOW. The row/column decoders, sense amplifiers, row buffers and bus drivers are all turned off to achieve maximal power reduction in the *precharge power-down* state.

3.3 DRAM Refresh Operation

Due to the limitation of materials, the charges stored in capacitors of DRAMs leak gradually over time. Over the span of the data retention times of DRAM cells, the voltages of the capacitors are lower than the threshold voltage and the stored data can not be read

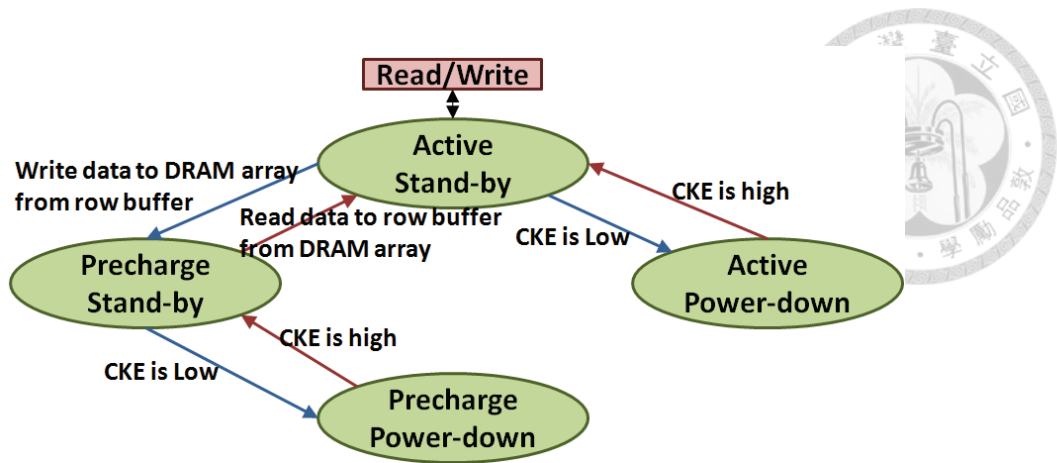


Figure 3.2: The power state transition of DDRx-SDRAM.

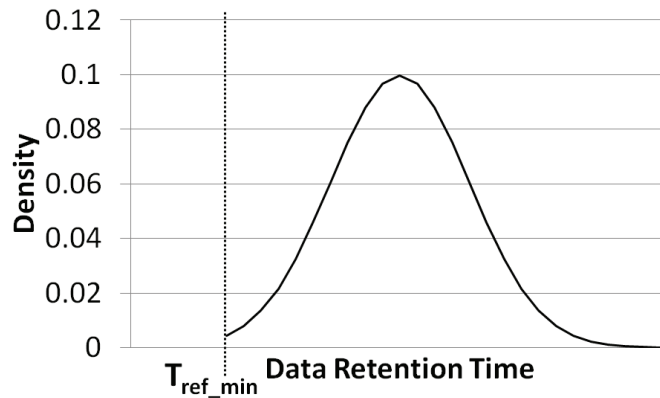


Figure 3.3: Data retention time distribution of DRAM cells.

anymore. Therefore, a periodic refresh of DRAM cells is necessary to guarantee data correctness. The refresh operation reads a row into the row buffer and write it back to recharge all DRAM cells of the row. Therefore, any data access to the bank that is in the refresh operation must be postponed until the refresh operation completes.

The refresh rate is typically set to be higher than the leakage rate of the fastest-leaking DRAM cells [78]. For example, with the distribution of data retention times of DRAM cells shown in Figure 3.3, the refresh interval is set as T_{ref_min} indicated by the dotted line, which is the shortest data retention time of all DRAM cells. In a conventional DRAM module, the refresh interval is usually set to 64ms. However, frequent refresh operations may cause high energy and performance overheads since all rows of a DRAM bank are read into the row buffer and then written back to recharge all the DRAM cells of the rows when the refresh operation is triggered for the bank. Setting the refresh interval

according to the worst case cell is over-provisioned for most DRAM cells. Kim et al. [122] show that only $10^{-6}\%$ of the cells have data retention times shorter than 128ms, and $10^{-4}\%$ of the cells have data retention times shorter than 500ms. Therefore, if I can correct the retention errors of DRAM cells with high leakage rate, I can utilize a refresh interval that is longer than their data retention times to reduce refresh power while data correctness is guaranteed.



Chapter 4

PPT Framework: DRAM Peripheral Leakage Reduction with Thermal Control

4.1 Introduction

As mentioned in Chapter 1, due to the interplay among performance, power and thermal, a technique optimized for one factor only often causes uncontrollable degradation at other design metrics. To tackle this issue, in this dissertation, I propose the first joint performance, power and thermal (PPT) management framework for DRAM memory in multi-core systems. PPT orchestrates task execution and page allocation to achieve desirable tradeoff between performance, power and temperature. In the PPT framework, both threads and memory resources are partitioned into groups. Threads of the same group are scheduled concurrently and only one group is active at each scheduling interval. Memory modules allocated to nonactive groups are put into the low-power mode to save energy. From thermal aspect, alternating active groups periodically allows memory modules to cool down in their idle periods. The main challenge in the proposed PPT framework is to determine how much memory resources should be allocated to a group to sustain the bandwidth demand of current workloads. To this end, I propose an *Adaptive Grouping* mechanism which dynamically adjusts PPT configuration (i.e., number of groups, and resources per group) to satisfy the bandwidth demand while minimizing power consumption.

To evaluate the proposed PPT framework, I compare the PPT framework with two task scheduling/page allocation policies, performance-driven and power-driven policies. Both policies schedule threads in round-robin fashion. The performance-driven policy distributes pages to all memory modules, and the power-driven policy allocates pages in the order they are accessed. The experimental results show that compared to

the performance-driven policy, PPT with the proposed dynamic bandwidth estimation method reduces power consumption by 23.8% and delivers comparable throughput; while the power-driven policy achieves 25.9% power reduction at the cost of peak throughput reduction of over 50% (9% on the average). From the thermal aspect, PPT has the lowest peak temperature. The peak temperature of the DRAM subsystem with the power-driven policy is 10.5°C higher than that with PPT.

In this chapter, Section 4.2 describes the details of the proposed PPT framework, and Section 4.3 shows the evaluations for the PPT framework.

4.2 The PPT Framework

The main idea of the proposed PPT framework for reducing DRAM peripheral leakage power and managing DRAM operating temperature is to orchestrate task execution and page allocation to achieve desirable tradeoff between performance, power and temperature. I use throughput as the measurement of memory system performance. Figure 4.1 shows one PPT configuration for a 4-channel memory system where each channel contains two ranks. In this example, threads and ranks are partitioned into 4 groups. Each group uses two ranks associated with two different channels. Threads of the same group are scheduled simultaneously. Threads in different groups are scheduled in round-robin fashion. Figure 4.2 illustrates the effect of group scheduling on power and thermal behavior. Only one group is active at each scheduling interval. The memory ranks of non-active groups could be turned into the low-power mode to save energy. Alternating active groups periodically allows memory ranks to cool down in their idle periods as shown in Figure 4.2(b).

Different PPT configurations affect the power and performance of the memory

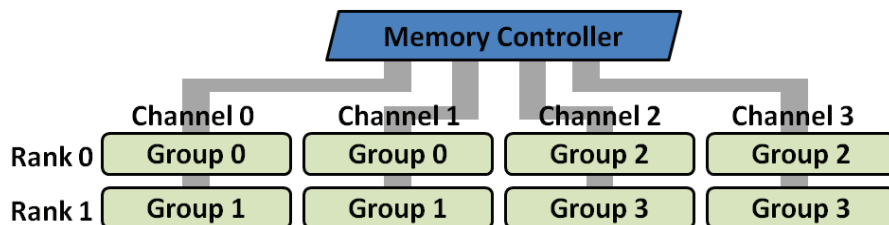


Figure 4.1: 4-group PPT configuration.

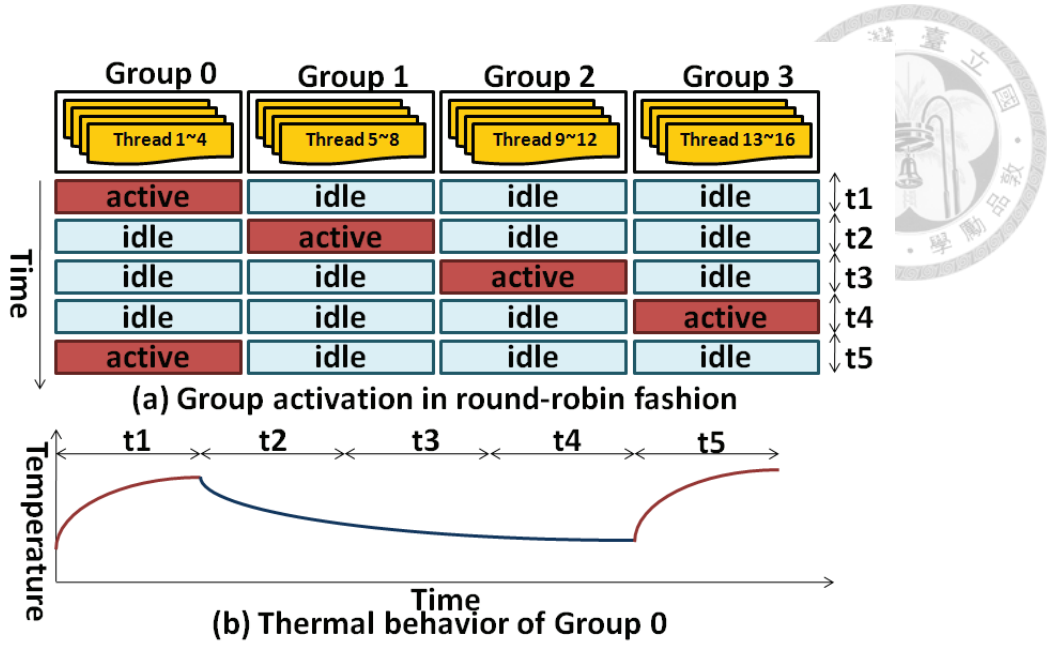


Figure 4.2: Group scheduling of the PPT framework.

system. When the memory system is partitioned into more groups, ranks and channels allocated to each group are fewer; that means more memory ranks could be shut down to achieve more power savings, which in turn reduces available bandwidth. To achieve desirable tradeoff between performance and power, I would like to partition threads into as many groups as possible provided that the allocated channels are enough to sustain the bandwidth demand of running threads. To adapt to dynamic system loading, the proposed PPT framework implements an *Adaptive Grouping* mechanism that monitors system loading to decide the appropriate PPT configuration to achieve power savings and meet the performance demand at the same time. Next, I describe the details of the *Adaptive Grouping* mechanism, and how thermal control is performed in PPT.

4.2.1 Adaptive Grouping Mechanism

A PPT configuration is represented as $\{G, C, R\}$, where G is the number of groups, C is the number of channels allocated to a group, and R is the number of ranks per group. For a memory system with m channels, and n ranks per channel, the possible group numbers are the factors of $m \cdot n$. For example, when $m = 4$ and $n = 2$, the possible group numbers are 1, 2, 4 and 8. I represent the possible group numbers as a set $S = \{g_1, g_2, \dots, g_k\}$, where $g_i < g_{i+1}$ ($i \in \{1 \dots k-1\}$). For a PPT configuration $\{g, c, r\}$, the available threads

in system (t) are partitioned into g groups, and ranks of a group are distributed among c channels.

The objective of *Adaptive Grouping* is to adjust the PPT configuration dynamically to adapt to the variation of system loading such that the memory resources allocated to a group are sufficient for the bandwidth demand of the group and the number of ranks per group is minimized. *Adaptive Grouping* includes two steps. First, I select the new PPT configuration according to the system loading. Second, I apply the new PPT configuration and perform page migrations, if necessary.

4.2.1.1 PPT Configuration Selection

The major challenge to decide the new PPT configuration is to accurately estimate the bandwidth demand of concurrently running threads. In this dissertation, I propose two approaches: the static and dynamic bandwidth (BW) estimation methods. The static BW estimation method relies on off-line profiling to obtain the average bandwidth demand for a thread, while the dynamic BW estimation method monitors the usage of the request buffer in the memory controller to adjust the PPT configuration accordingly.

4.2.1.1.1 Static Bandwidth (BW) Estimation Method To estimate system bandwidth demand (BD), I assume that the average bandwidth requirement of a thread is obtained through off-line profiling¹. In order not to underestimate the system bandwidth demand, the maximal total bandwidth requirement of threads that could be scheduled concurrently is used for system bandwidth estimation. Therefore, the bandwidth demand $BD(g)$ in a system is obtained with the following formula:

$$BD(g) = MAX_{i=1}^g \left(\sum_{j=1}^{core} t_{i,j} \right), \quad (4.1)$$

where g is the number of groups and $t_{i,j}$ is the bandwidth demand of the j^{th} high bandwidth thread in i^{th} group, and $core$ is the number of cores in the system. Therefore, to minimize $BD(g)$, threads are partitioned to achieve balanced bandwidth demand among groups.

¹The bandwidth demand of a thread is the summation of page fault traffic and L2 miss traffic divided by the execution time.

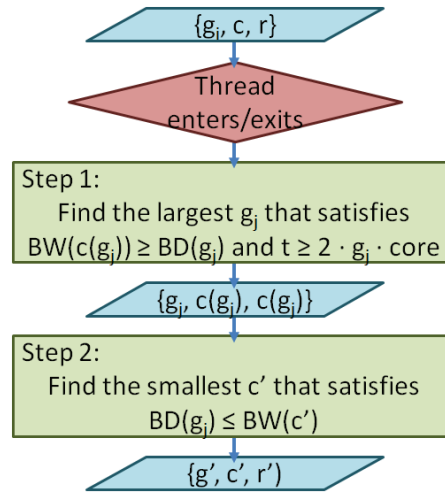


Figure 4.3: Static BW estimation method for Adaptive Grouping in the PPT framework.

In addition to bandwidth consideration, the number of threads in the system should also be taken into account when choosing the PPT configuration. Since only threads of the same group can be scheduled simultaneously, in order to fully utilize available cores in the presence of long latency operations, such as page faults, the number of threads in each group should be at least $2 \cdot core$, where $core$ is the number of cores in the system.

In the static BW estimation method, *Adaptive Grouping* is triggered when a thread enters or exits the system. Figure 4.3 shows the steps of the proposed static BW estimation method. In the first step, I select the largest g that satisfies both the bandwidth and thread number constraints. Due to the thread number constraint, I may be forced to select a smaller g than required; that is the bandwidth support of the channels allocated to a group is more than the bandwidth demand. In this case, the second step is invoked to reduce the number of channels and ranks allocated to each group. Below I detail these two steps assuming the PPT configuration is $\{g_i, c, r\}$ before *Adaptive Grouping* is triggered.

Step1: Examine all possible g and find the largest g that satisfies the following equation.

$$BW(c(g)) \geq BD(g) \text{ AND } t \geq 2 \cdot g \cdot core, \quad (4.2)$$

where $c(g)$ is the number of channels which can be allocated to a group when threads are partitioned into g groups, and $BW(c(g))$ is the bandwidth of $c(g)$ channels.

If *Adaptive Grouping* is triggered with a new thread, I need to examine $\{g_1 \sim g_{i+1}\}$, i.e., all the group numbers smaller than g_i due to higher system bandwidth, and one level higher than g_i because I may be able to partition threads into finer granularity due to more threads in the system now. Similarly, if *Adaptive Grouping* is triggered because a thread exits the system, the possible g is $\{g_{i-1} \sim g_k\}$. Given g , the number of channels which can be allocated to a group ($c(g)$) is given by the following formula:

$$c(g) = \begin{cases} \frac{m \cdot n}{g} & (g \geq n) \\ m & (g < n) \end{cases} \quad (4.3)$$

For a memory system with m channels and n ranks per channel, when I partition the memory system into g groups, each group has at most $\frac{m \cdot n}{g}$ ranks. To maximize the number of channels allocated to a group, the $\frac{m \cdot n}{g}$ ranks of a group are distributed to as many channels as possible. For example, in a 4-channel, 2-rank per channel system, if g is 2, each group can access 4 ranks ($\frac{4 \cdot 2}{2}$), and these ranks are distributed to 4 different channels. Therefore, $c(2) = 4$. In this case, when $g < n$, then each group is allocated more than m ranks; that means each group could access all channels, therefore, $c(g) = m$. Since I want to minimize the number of ranks per group while providing sufficient bandwidth, the number of ranks per group is set as $c(g)$. Now, I have a new PPT configuration, $\{g_j, c(g_j), r(g_j)\}$, where $r(g_j) = c(g_j)$.

Step2: In this step, I examine if I could reduce the channels and ranks allocated to a group. For example, in a 4-core system with only 7 threads, I am not able to partition these threads into multiple groups. Therefore, in the first step, g_j is equal to 1, and $c(g_j)$ is the total number of channels in the system. If $BD(g_j) < BW(c(g_j))$, then I could reduce the channels and ranks allocated to this group. Therefore, in the second step, I decide the minimal amount of channels c' to satisfy $BD(g_j) \leq BW(c')$. Therefore, in this step, I get a new PPT configuration $\{g', c', r'\}$, where $g' = g_j$ and $r' = c'$.

4.2.1.1.2 Dynamic Bandwidth (BW) Estimation Method In the dynamic BW estimation method, the usage of the request buffer in the memory controller is treated as an indication whether the current memory resources are under-provisioned or over-provisioned. In the memory controller, the request buffer is used to store pending memory requests. So if the request buffer is usually full, it means memory requests arrive more frequently than

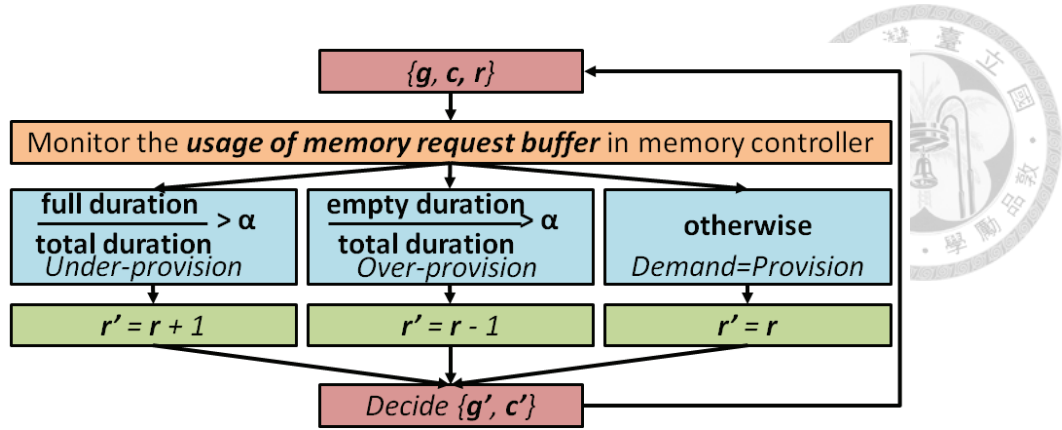


Figure 4.4: Dynamic BW estimation method for Adaptive Grouping in the PPT framework.

the current DRAM resources can service (i.e., request arrival rate $>$ service rate). In contrast, if the request arrival rate is lower than the service rate, the request buffer is usually empty. Therefore, in the dynamic BW estimation method, I monitor the usage of the request buffer in the memory controller and adjust the PPT configuration accordingly.

Figure 4.4 illustrates the dynamic BW estimation method. I monitor the usage of the request buffer periodically with two metrics: full-ratio and empty-ratio. Full-ratio is the percentage of time when the request buffer is full, while empty-ratio indicates the percentage of time when the request buffer is empty. If full-ratio/empty-ratio reaches a predefined threshold, α , I increase/decrease the number of ranks allocated to each group by one. The α value needs to be carefully selected. If α is too small, the system will be too sensitive to system variation thereby incurring frequent PPT configuration change. On the other hand, if α is too large, the system will be slow to respond to variations of system loading. Note that the value of α should be larger than 50%, such that only one of the under-provision and over-provision conditions can hold.

With the new rank configuration, I now describe how to decide the number of channels (c) and groups (g) in the new PPT configuration. Recall that ranks are distributed to all channels. Therefore, if r' is not greater than the total number of channels in the system, then $c' = r'$, otherwise, $c' = m$ in an m -channel memory system. With new c' and r' , I then decide the appropriate group number. For a memory system with m channels and n ranks per channel, the total number of ranks in the system is $m \cdot n$. Since each group contains r' ranks in the new PPT configuration, there should be $\lfloor \frac{m \cdot n}{r'} \rfloor$ groups at most.

In addition to the resource consideration, as mentioned previously in Section 4.2.1.1.1, I need to consider the thread number constraint. That is, each group must have at least $2 \cdot core$ threads, where $core$ is the number of cores in the system. Therefore, to decide the new group number, I select the largest g' that satisfies the following condition:

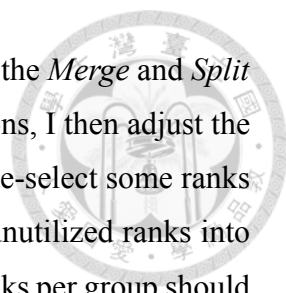
$$g' \leq \frac{m \cdot n}{r'} \text{ AND } g' \leq \frac{t}{2 \cdot core}. \quad (4.4)$$

Note that, due to the thread number constraint, the group configuration (i.e., g') needs to be re-examined every period regardless if the rank value changes or not.

The dynamic BW estimation method overcomes three weaknesses of the static BW estimation approach. First, the static BW estimation method only considers the average bandwidth demand of each thread, while the dynamic BW estimation method can capture bandwidth variation within each thread execution. Second, the static BW estimation method calculates the system bandwidth demand by considering the maximal total bandwidth requirement of threads which could be scheduled concurrently. Therefore, the static BW estimation method tends to overestimate the bandwidth demand of workloads. Third, the static BW estimation method is not able to estimate the bandwidth demand of threads that are swapped out due to some long latency operations, e.g. page faults. I call these memory activities as the background memory activities that are issued by DMA engines or other I/O operations but not the scheduled threads on processors. So if the workloads are triggering excessive background memory activities, the static BW estimation method may underestimate the system loading.

4.2.1.2 New PPT Configuration Adoption

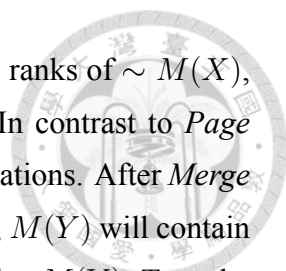
To apply the new PPT configuration $\{g', c', r'\}$ to the system, I first re-partition threads into g' groups. If $g' < g$, I perform *Merge* which merges threads in two different groups into one group. Let $M(X)$ represent the memory ranks which can be used by group X , $\sim M(X)$ represent the memory ranks which should not be used by group X , and $T(X)$ represent the threads assigned to group X . After *Merge* $(X, Y) \rightarrow (Z)$, $T(X)$ and $T(Y)$ could allocate pages on both $M(X)$ and $M(Y)$. *Merge* operations are performed repeatedly until $g' = g$. If $g' > g$, a *Split* operation is performed. After *Split* $(Z) \rightarrow (X, Y)$, $T(Z)$ is partitioned into $T(X)$ and $T(Y)$ with balanced bandwidth between groups in mind. After a *Split* operation, $T(X)(T(Y))$ could only allocate pages on $M(X)(M(Y))$.



Split operations are also performed repeatedly until $g' = g$. Note that the *Merge* and *Split* operations are performed on all groups. After *Merge* or *Split* operations, I then adjust the resources allocated to each group according to c' and r' . If $r' < r$, I de-select some ranks from $M(X)$ to reduce the number of $M(X)$. If $r' > r$, I add some unutilized ranks into $M(X)$ to increase the number of $M(X)$. After that, the number of ranks per group should be r' .

Page migration may be required to make the access pattern conform to the new PPT configuration. For example, after *Split* (Z) \rightarrow (X, Y), $T(X)$ may still access $M(Y)$. If most of the memory accesses of $T(X)$ do not map to $M(X)$, the new PPT configuration does not take any effect. Page migration should be managed carefully since it incurs both performance and energy overheads. I propose a *Deferred Page Migration* approach to exploit the temporal locality of memory accesses. That is, after the PPT configuration changes, I do not move pages immediately, because running threads might bring in new pages, which are very likely to be hot data due to temporal locality. Therefore, to decide whether to perform a page migration, I monitor how the accesses are distributed among ranks in the new configuration. If many memory accesses of $T(X)$ are not mapped to $M(X)$, I will try to move the pages of $T(X)$ from $\sim M(X)$ to $M(X)$, and the operation is called *Page Concentration*. On the other hand, if the memory accesses in $M(X)$ are not distributed evenly, I will try to balance the loading among $M(X)$, and the operation is called *Page Distribution*.

Page Concentration may be triggered due to *Split* operations. As I mentioned in the previous paragraph, after *Split* (Z) \rightarrow (X, Y), if most of the memory accesses of $T(X)$ do not map to $M(X)$, the *Split* operation does not take any effect. In this case, *Page Concentration* should be triggered to make the access pattern conform to the new PPT configuration. Since I want to map all memory accesses of $T(X)$ to $M(X)$, I monitor the access distribution between $M(X)$ and $\sim M(X)$, and *Page Concentration* is triggered when the ratio of accesses mapped to $\sim M(X)$ reaches a predefined *Concentration Threshold*. For each *Page Concentration*, I move n recently accessed MRU pages of $T(X)$ to $M(X)$ from $\sim M(X)$, where n is a small number to limit the migration overhead. Besides *Split* operations, reducing the number of ranks per group may also trigger *Page Concentration* to move MRU pages of $T(X)$ from $\sim M(X)$ to $M(X)$ in the same



way, because the memory accesses of $T(X)$ may be mapped to some ranks of $\sim M(X)$, which belonged to $M(X)$ before the PPT configuration changes. In contrast to *Page Concentration*, *Page Distribution* may be triggered due to *Merge* operations. After *Merge* $(X, Y) \rightarrow (Z)$, if there are no new pages of $T(X)$ allocated to $M(Y)$, $M(Y)$ will contain no pages belonging to $T(X)$, and no accesses of $T(X)$ will be mapped to $M(Y)$. To make the access pattern conform to the new PPT configuration and fully utilize the bandwidth of $M(Y)$ to sustain the system bandwidth demand, I want to map all memory accesses of $T(X)$ to the ranks in both $M(X)$ and $M(Y)$ evenly. So, I check if the memory accesses of $T(Z)$ (including $T(X)$ and $T(Y)$) are evenly distributed to all ranks of $M(Z)$. When accesses are evenly distributed, the numbers of accesses mapped to the hottest rank (i.e., the most frequently accessed rank) and the coldest rank (i.e., the least frequently accessed rank) should be the same. Therefore, *Page Distribution* is triggered when the access ratio between the hottest rank and the coldest rank reaches a predefined *Distribution Threshold*. For each *Page Distribution*, I move n MRU pages of $T(Z)$ from the hottest rank to the coldest rank. In addition to *Merge* operations, increasing the number of available ranks for $T(X)$ may also trigger *Page Distribution* to move some MRU pages of $T(X)$ to the newly allocated ranks in $M(X)$, because the newly allocated ranks may contain no pages belonging to $T(X)$ and no accesses of $T(X)$ are mapped to these ranks. Please note that, I perform page migrations for all groups in the same way.

4.2.1.3 Architectural Support

The PPT framework requires two architectural supports. For the dynamic BW estimation method, since it is a reactive approach to decide PPT configurations according to the usage of the request buffer in the memory controller, I need two hardware counters: *full-counter* and *empty-counter* in the memory controller. The *full-counter* is increased by 1 when the request buffer is full at this cycle, and the *empty-counter* is increased by 1 when the request buffer is empty. I also need a memory mapping interface to read and reset the hardware counters periodically. Based on the readings of the hardware counters, the operating system can perform the dynamic BW estimation method accordingly.

Note that, the static BW estimation method does not require special architectural support, since all required information can be obtained from the operating system and conventional performance monitor units. In the static BW estimation method, to obtain the

average bandwidth requirement of a thread, the execution time, the page fault traffic and the L2 miss traffic of a thread are required. The execution time and the number of page faults of a thread can be easily obtained from the operating system, since the operating system is aware of thread execution and virtual memory management. The number of L2 misses can be read from conventional performance monitor units. The remaining calculation of average bandwidth requirement of a thread can be done in the operating system, which records all necessary information.

For *Deferred Page Migration*, one hardware counter for each rank is required to record the number of accesses in the rank. If the page migrations are required, DMA-like operations are performed to move the pages and the operating system is interrupted to update the page table.

4.2.2 Thermal Control Policy

Thermal control in the PPT framework is achieved through *Group Switching* and *Activity Migration*. When the system has enough threads for partitioning into groups, balanced temperature among ranks could be achieved through periodically scheduling threads in different groups as illustrated in Figure 4.2. The question remains to be answered is how long the group switch interval should be. To achieve power saving and prevent memory ranks from overheating, the group switch interval needs to be long enough to transit idle ranks into a low-power mode, and short enough to balance the temperature among memory ranks. When a rank is idle, according to the power management policy of DDRx, it takes several microseconds to perform a refresh operation and enter the precharge power-down state, which is the state that consumes the least power. Therefore, the group switch interval should be longer than several microseconds to transit the rank into the precharge power-down state. As for thermal control, I stressed the memory system to see how fast temperature rises. I found that with the heaviest loading, the operating temperature of a DDR2-SDRAM chip only increases at most 1°C in about 40 milliseconds². Therefore, I know that if group switch interval is shorter than 40 milliseconds, temperature could be kept balanced among ranks to avoid hotspot. In modern operating systems, the context switch interval is usually set to 1 ~ 10 ms, which is much longer than several microsec-

²I use the simulation setup described in Section 4.3.1 to perform this experiment.

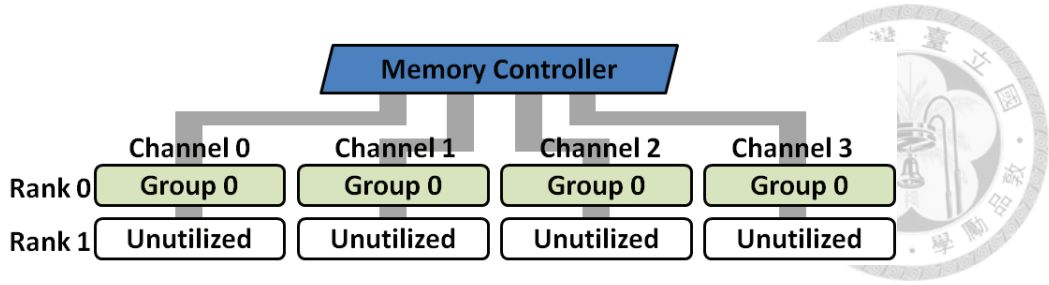


Figure 4.5: 1-group PPT configuration.

onds, and shorter than 40 milliseconds. Therefore, setting group switch interval equal to context switch interval could achieve power savings while maintaining balanced temperature at the same time without causing extra OS scheduling overheads.

When there are not enough threads for partitioning into groups, I use *Activity Migration* to control temperature. In this case, threads in a group may use only one rank per channel, and there may exist ranks which are not utilized at all. For example, for a $\{1, 4, 4\}$ configuration as illustrated in Figure 4.5, *rank1* of all channels are not allocated to any group. These unutilized ranks could be used for thermal control. When the temperature of *rank0* in *channel0* exceeds the thermal threshold, *rank0* in *channel0* is de-selected, and *rank1* in *channel0* is selected to replace *rank0*. I call this *Activity Migration*. Since old pages of the group are still allocated in *rank0* in *channel0*, page migration may be necessary to achieve desired temperature behavior. Here, I adopt the *Deferred Page Migration* approach described in Section 4.2.1.2. Please note that *Activity Migration* could be also used together with *Group Switching* for thermal control as long as there are unutilized ranks.

4.3 Evaluation to PPT Framework

4.3.1 Experimental Setup

I evaluate the proposed PPT framework with trace-driven simulation. The virtual memory address traces are obtained by Cachegrind, which is the cache simulation component of Valgrind [123] profiling tool. I use the cycle-accurate DRAM simulator, DRAMsim [124], to model the DRAM system. The traces from different programs are combined together to form multi-core workloads. I annotate the memory traces with correct physical addresses and timestamps according to the page allocation and thread scheduling policies.



Table 4.1: Processor and memory configurations for PPT evaluation.

Parameters	Value
Processor	4-core, 2GHz, 1 hardware context per core
L1 caches (per core)	64KB Inst/64KB Data, 2-way, 64B line
L2 caches (per core)	1MB, 8-way, 64B line
Memory	4 channels, 2 DIMMs/channel, 1 rank/DIMM
Memory controller	32-entry request buffer
Memory rank	1GB, 800MHz DDR2-SDRAM, 32 banks/rank
Memory bank	8192 rows/bank, 512 columns/bank
Channel bandwidth	8-byte/channel, 6.4GB/s

I also model page faults and their associated memory accesses, and reflect the delay caused by memory contention, thread scheduling, and page faults by adjusting the timestamps in traces. To evaluate DRAM temperature, I adopt HotSpot 3.0 [125], which calculates temperature by modeling DRAM device physical properties.

Table 4.1 shows the processor and memory configurations. The detailed timing and power configurations of DRAM chips are obtained from Micron’s DDR2-SDRAM MT47H64M8CF-25E [126]. I assume the open-page row buffer management policy. The refresh interval of each row is set to 64 milliseconds. I adopt single rank DDR2-SDRAM DIMMs, which are commonly used. Therefore, for thermal simulation, there is no thermal interaction between any two ranks, because the ranks are far from each other. The thermal resistance is set to 5K/W and the ambient temperature is set to 45°C.

For the proposed PPT method, both the static and dynamic BW estimation methods are evaluated. In the dynamic BW estimation method, the α threshold value is set to 70%. In *Deferred Page Migration*, *Concentration Threshold* is set to 0.1 and *Distribution Threshold* is set to 2. The monitoring interval of *Deferred Page Migration* is 1 millisecond. For each page migration, I move 25 pages. The temperature threshold for triggering the *Activity Migration* mechanism described in Section 4.2.2 is 84°C to make the peak temperature of memory under 85°C. The group switch interval is equal to the context switch interval, which is 1 millisecond in these experiments. Besides the proposed PPT methods, I also implement two other page allocation policies, performance-driven and power-driven policies. The performance-driven policy distributes pages to all ranks. To avoid contention, the performance-driven policy balances the loading between ranks based on the number of page faults of ranks. The power-driven policy allocates pages in the order they accessed, and an entire rank is filled up before moving to the next, such that the accessed ranks are minimized. It is the Sequential First-Touch policy in [10].

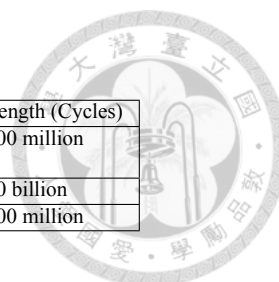
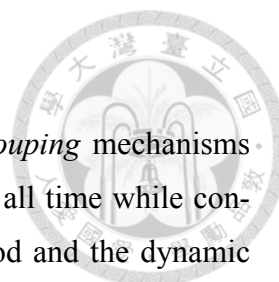


Table 4.2: Workloads for PPT evaluation.

Workload	Benchmarks	Length (Cycles)
SPEC2000-1	{177.mesa, 181.mcf, 164.gzip, 255.vortex, 197.parser, 173.applu, 176.gcc, 256.bzip2, 301.apsi, 179.art, 171.swim, 183.quake} \times 8	800 million
SPEC2000-2	171.swim, 173.applu, 176.gcc, 183.quake, 256.bzip2, 301.apsi	20 billion
SPECjbb	light-loading SPECjbb \times 64, heavy-loading SPECjbb \times 64	800 million

Both policies adopt LRU as the replacement policy, and work with the round-robin thread scheduling policy.

The workloads tested for the PPT framework are from SPEC CPU2000 and SPECjbb2005 [127]. The detailed information of workloads are listed in Table 4.2. SPEC2000-1 and SPEC2000-2 are composed of SPEC CPU2000 benchmarks. SPEC2000-1 simulates a system with thread number ranging from 1 to 88 over time. In this workload, I have enough threads to partition threads into groups. I vary the system loading by creating new threads periodically (8 new threads are created every 0.5 second), and terminating a thread after it executes 800 million cycles. SPEC2000-2 is created to test how PPT performs when the system does not have enough threads for grouping. Therefore, SPEC2000-2 includes only 6 threads. In SPEC2000-2, applications run longer than SPEC2000-1 so the peak temperature could be higher than the threshold temperature (84°C) to trigger *Activity Migration*. I create the SPECjbb workload to test how PPT performs for transaction-based server workloads. SPECjbb consists of threads from the SPECjbb2005 benchmark, which has much more page faults as I/O operations than the SPEC CPU2000 benchmark. A thread in the SPECjbb2005 benchmark represents an active user posting transaction requests within a warehouse. There is a one-to-one mapping between warehouses and threads. The SPECjbb2005 benchmark is inspired by the TPC-C benchmark and loosely follows the TPC-C specification, and the TPC-C benchmark usually includes thousands of warehouses in a database [128, 129]. Therefore, the SPECjbb workload could include thousands of threads to stress the memory system. However, due to the limitation of simulation time, I include 128 threads in SPECjbb. I vary the system loading of SPECjbb by creating light-loading threads and heavy-loading threads. In a computer with Xeon E5320 (1.86GHz), the throughput of a full speed heavy-loading thread is about 18000 bops, and that of a light-loading thread is about 500 bops. I first schedule the light-loading threads, and then schedule the heavy-loading ones, such that the system loading is light at the beginning and becomes heavy later.



4.3.2 Experimental Results

In this section, I first examine if the proposed *Adaptive Grouping* mechanisms achieve comparable throughput to the performance-driven policy at all time while consuming less power. I also compare the static BW estimation method and the dynamic BW estimation method in terms of throughput and power consumption. I then show how the PPT scheme controls memory temperature through *Group Switching* and *Activity Migration*. Finally, I summarize performance, power and thermal of the PPT, performance-driven and power-driven schemes.

4.3.2.1 Power/Performance Evaluation of Adaptive Grouping

Figure 4.6 shows the memory throughput breakdown of various threads in SPEC2000-1 with the performance-driven policy. To evaluate the PPT framework under system loading variation, I schedule 8 new copies of a benchmark into the system in SPEC2000-1 every 0.5 second. The total throughput of the 8 copies of a benchmark is plotted together in Figure 4.6. Due to workload variation of SPEC2000-1, we can observe that the delivered throughput of the performance-driven policy varies. The peak throughput is 9.4GB/s at 1.25s, while the lowest throughput is 0.06GB/s at 0.5s. Since I schedule different benchmarks into the system at different times, the major contributors of the memory throughput vary. At 1.25s and 3.75s, the threads of 164.gzip and the threads of 256.bzip2 are respectively the major contributors of the memory throughput, because they are scheduled at 1s and 3.5s and bring a lot of compulsory page faults into the system. From 1.75s to 3.5s, the threads of 181.mcf are the major contributors of the memory throughput, and the threads of 301.apsi are the major contributors from 4s to 5.75s. The threads of 179.art show significant contribution in memory throughput from 5s to 10s, and the threads of 171.swim also incur a lot of memory throughput from 6.5s to 10s. We can observe that the system loading changes due to the change of executed threads and the change of the program behaviors of the benchmarks.

Figure 4.7 compares the memory throughput and power consumption of four policies for SPEC2000-1. The delivered throughput of both *Adaptive Grouping* mechanisms is close to that of the performance-driven policy at all time, while the power-driven policy suffers the most throughput degradation among four schemes. At the time when the

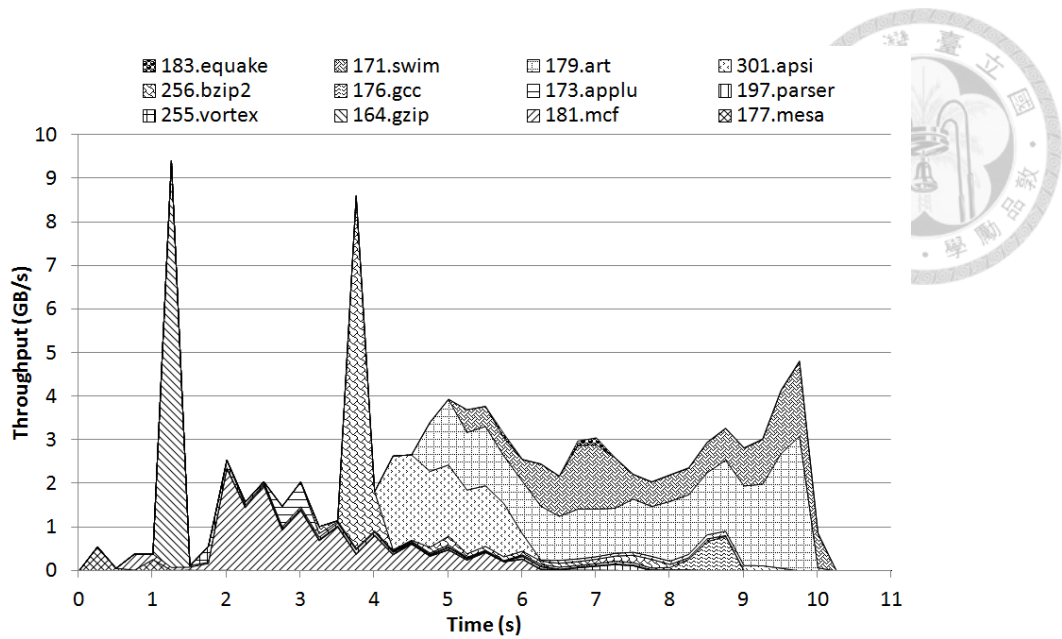


Figure 4.6: Memory throughput breakdown of threads in SPEC2000-1 with the performance-driven policy for PPT evaluation.

performance-driven and *Adaptive Grouping* mechanisms show peak throughput of 9.4GB/s, the power-driven scheme could only deliver 4.6GB/s. The performance-driven and *Adaptive Grouping* schemes finish serving all the requests in about 10 seconds, while the power-driven policy takes one more second. Note that the throughput of the power-driven policy is sometimes higher than other three policies. This is because in the power-driven policy, more requests are queued due to insufficient bandwidth and these requests are handled later when the system loading is light. From the power aspect, we can observe that the power consumptions of both *Adaptive Grouping* mechanisms are lower than that of the performance-driven policy most of the time. When the system loading is light, between 6s ~ 9s, both *Adaptive Grouping* mechanisms consume similar power to the power-driven policy. Because the dynamic BW estimation method can estimate the system bandwidth demand accurately, but the static BW estimation method tends to overestimate the bandwidth demand, PPT with the dynamic BW estimation method can reduce more power consumption (7% in average) than PPT with the static BW estimation method. In 6s ~ 8s, the PPT with the dynamic BW estimation method even saves more power than the power-driven policy.

I now discuss the relationship between resource allocation and system loading, and

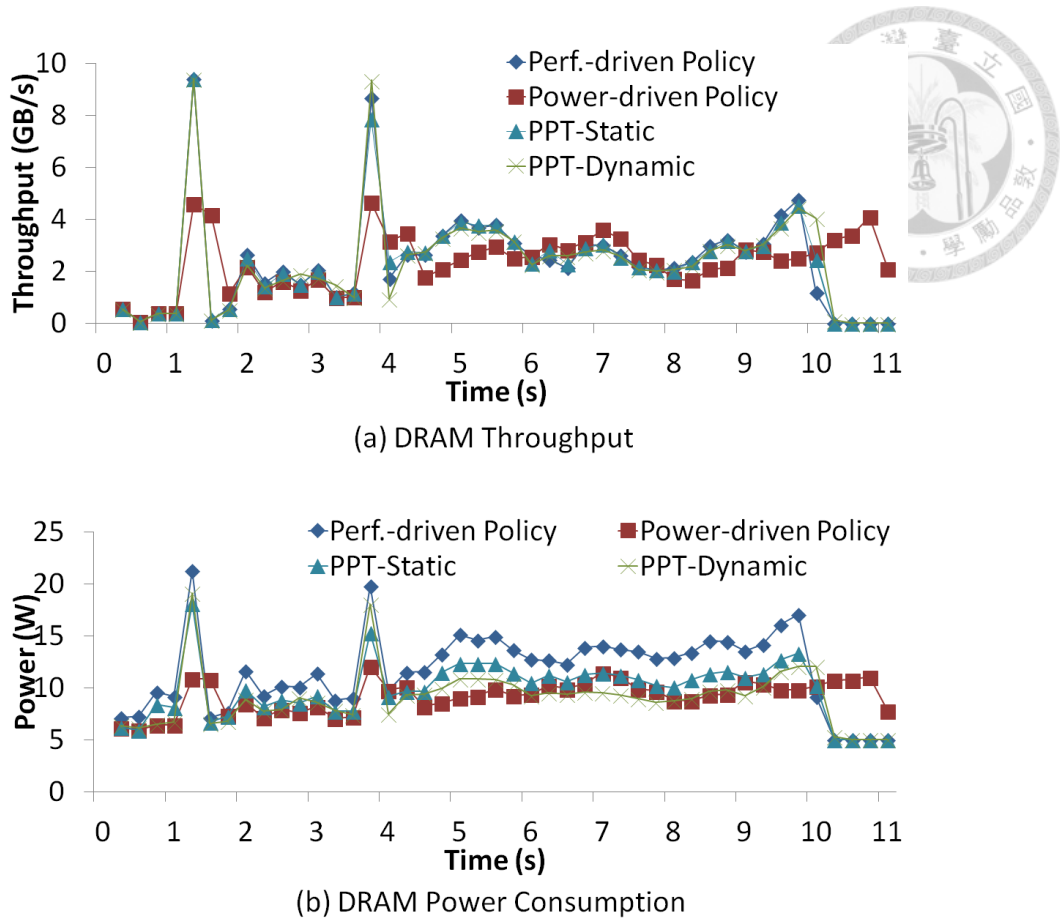


Figure 4.7: Throughput and power of DRAM system with SPEC2000-1 for PPT evaluation.

explain why the dynamic BW estimation method is better than the static BW estimation method and the power-driven policy. Figure 4.8 shows how the PPT scheme adjusts the resources allocated to each group (i.e., the number of ranks per group) to adapt to the workload variation, which is indicated as the memory system throughput of the performance-driven policy. I also show the average number of concurrently accessed ranks for the power-driven policy for comparison. We can observe that the number of ranks per group for the dynamic BW estimation method matches the throughput variation closely. For example, at 1.25s and 3.75s, the system shows peak throughput, and therefore the dynamic method allocates about 6 ranks and 4 ranks for a group, respectively. During 4.75s ~ 6s and 9.5s ~ 10s, the system loading is relatively heavy, so the dynamic method allocates about 2 ranks for a group. In the remaining durations, the system loading is light, and thus the dynamic method only allocates one rank for a group. As for the static BW es-

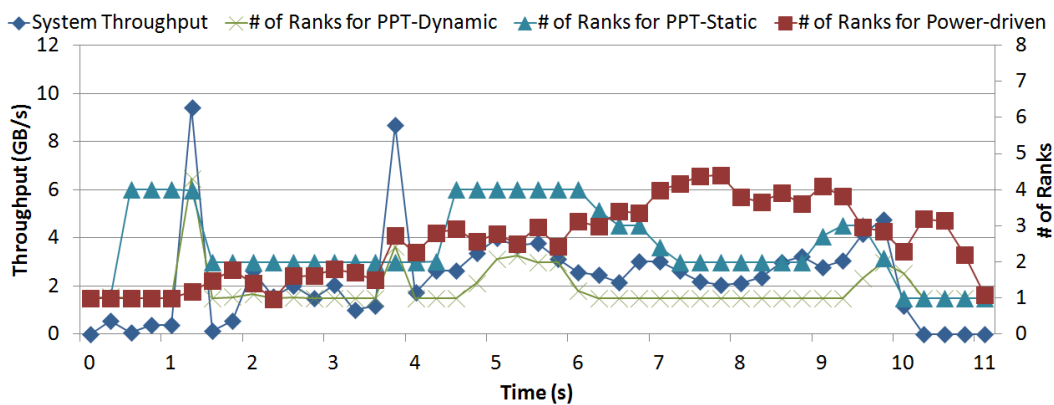


Figure 4.8: The throughput of the performance-driven policy, the number of ranks per group for the static and dynamic BW estimation methods, and the number of concurrently accessed ranks for the power-driven policy in SPEC2000-1 for PPT evaluation.

timination method, it often allocates more ranks per group than the dynamic method. The reason is that the static method takes the maximal total bandwidth demand of threads that could be scheduled concurrently as the system loading, so it often overestimates the system bandwidth demand. This is why the dynamic BW estimation method can achieve more power savings than the static one. In 6s ~ 8s, as shown in Figure 4.7(b), the dynamic BW estimation method saves more power than the power-driven policy. That is because, in 6s ~ 8s, the system loading is light, and the dynamic BW estimation method allocates one rank to sustain the bandwidth demand, but due to large footprint of SPEC2000-1, the power-driven policy allocates pages in five ranks. Since the scheduler in the power-driven policy is not aware of the page allocation, it schedules the threads accessing these ranks simultaneously as shown in Figure 4.8, and loses the opportunities to turn these ranks off.

Figure 4.9 shows the memory throughput breakdown of various threads in SPEC2000-2 with the performance-driven policy. Recall that SPEC2000-2 contains only 6 threads, so the system loading is very light, and all threads are all created at the beginning, so it incurs a peak bandwidth demand at the beginning due to compulsory page faults. From 0.5s to 3.5s, the threads are in a phase that has relatively low throughput, and then they are in another phase that has relatively high throughput from 3.5s to 15.5s. However, the system loading is still very light from 3.5s to 15.5s.

In Figure 4.10, I show the memory throughput and power consumption of the

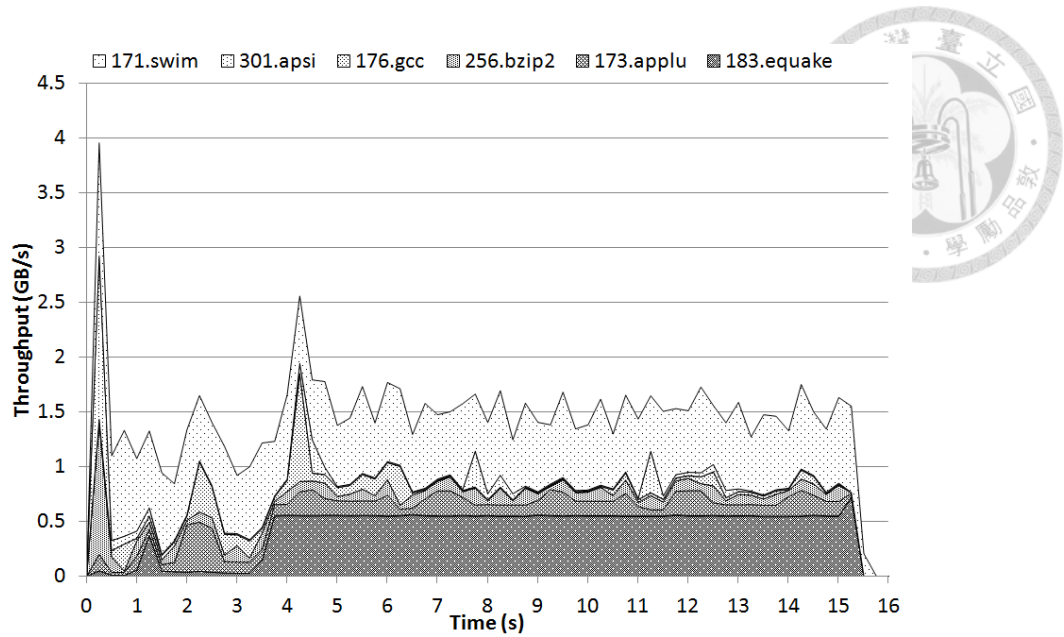
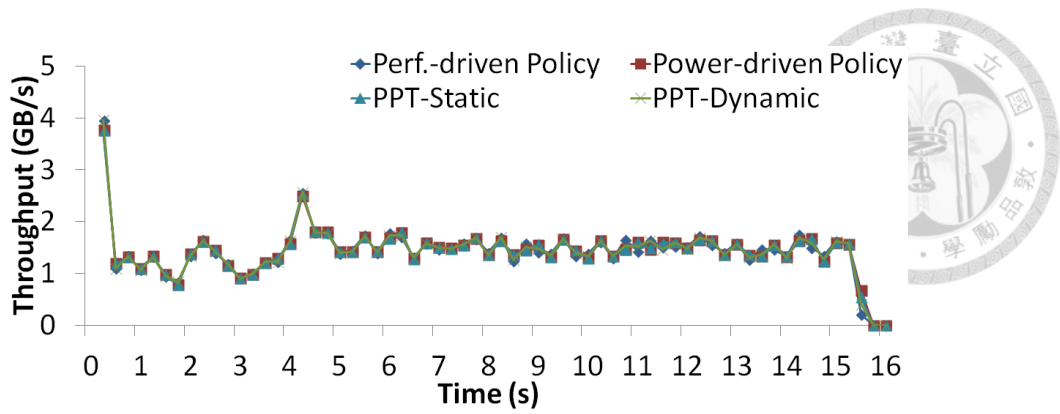
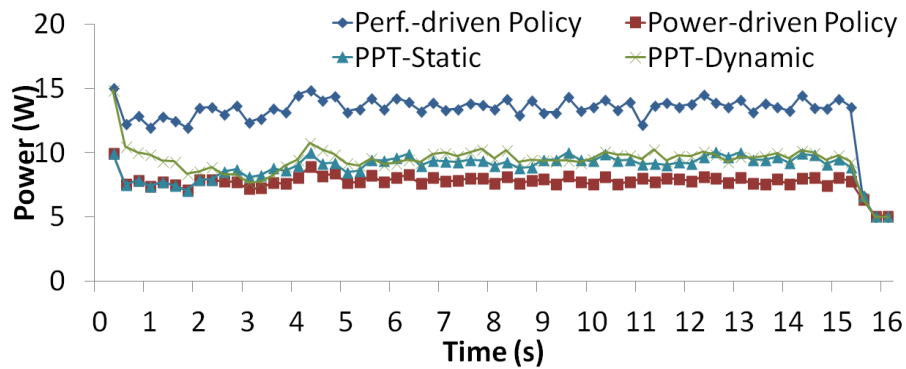


Figure 4.9: Memory throughput breakdown of threads in SPEC2000-2 with the performance-driven policy for PPT evaluation.

performance-driven, power-driven and *Adaptive Grouping* mechanisms for SPEC2000-2. We can observe that all policies deliver similar throughput except that the static BW estimation method and the power-driven policy show relatively low throughput at the beginning compared to other two policies. This is because they both allocate only one rank for the running threads. The static BW estimation method only considers the average bandwidth demand of each thread during entire program execution, so it underestimates the peak bandwidth demand at the beginning. Except for the beginning stage, the static BW estimation method and the power-driven policy also achieve the same level of performance as the performance-driven policy in the steady state, since the system loading is very light. As for the power consumption, the dynamic method consumes more power than the static one in the initial phase, but it gradually approaches to the static one. After the initial phase, most of the pages are brought in, therefore, the system bandwidth demand drops as shown in Figure 4.10(a). The dynamic scheme successfully detects the change and allocates only one rank to the group afterwards. However, even though there is only one rank per group for the PPT schemes, the results show that they still have higher power consumption than the power-driven policy. It is because *Activity Migration* is triggered for thermal control and associated page migrations incur power overheads. I will discuss the overheads for thermal control in Section 4.3.2.4.



(a) DRAM Throughput



(b) DRAM Power Consumption

Figure 4.10: Throughput and power of DRAM system with SPEC2000-2 for PPT evaluation.

In Figure 4.11, I show the memory throughput breakdown of the threads in SPECjbb with the performance-driven policy. The total memory throughput of all 64 light-loading threads is plotted together as SPECjbb-Light, and the memory throughput of other 64 heavy-loading threads is plotted together as SPECjbb-heavy. We can see that both light-loading threads and heavy-loading threads incur massive compulsory page faults at the beginning of the thread execution. The light-loading threads contribute about 0.44GB/s at the steady state and the heavy-loading threads contribute about 2.78GB/s at the steady state.

Figure 4.12 compares the memory throughput and power consumption of four policies for SPECjbb. The most different workload behavior of SPECjbb to SPEC2000-1 and SPEC2000-2 is that SPECjbb has many I/O operations incurred by page faults. These I/O operations are served by DMA in background. Since, in the static BW esti-

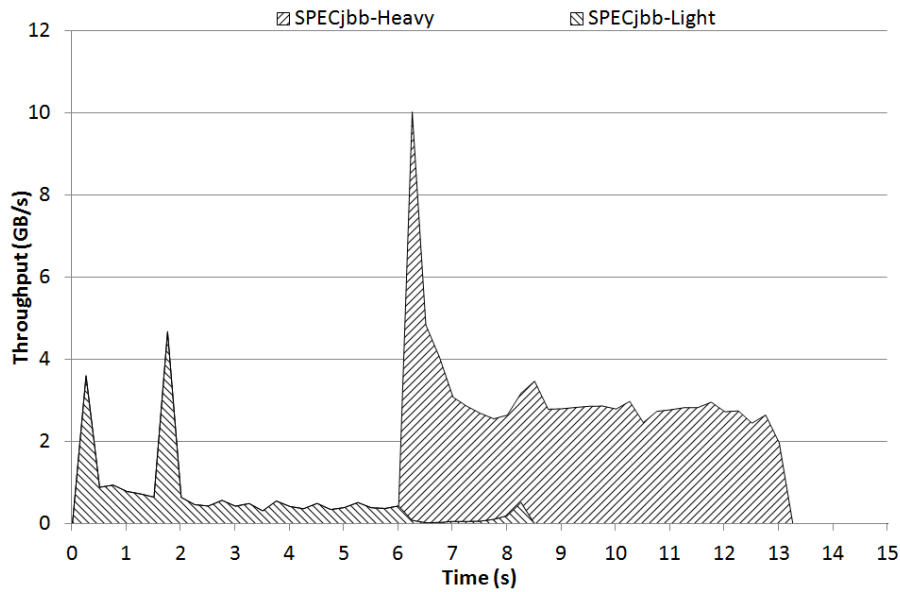


Figure 4.11: Memory throughput breakdown of threads in SPECjbb with the performance-driven policy for PPT evaluation.

mation method, the system bandwidth demand is obtained by considering the maximal total bandwidth requirement of running threads, the static method does not consider the memory accesses from I/O operations of idle threads which are swapped out due to page faults. In this case, the static method often underestimates the system bandwidth demand in the SPECjbb workload, since many I/O operations in SPECjbb are not taken into account. Therefore, from Figure 4.12(a), we can see that the static method is not able to achieve similar throughput as the performance-driven scheme after 6s, because there is a peak bandwidth demand due to many compulsory page faults from newly scheduled heavy threads. While for the dynamic method, we can see that it still achieves throughput close to the performance-driven policy, and its power savings is similar to the power-driven policy. Since the static method underestimates the bandwidth demand and only allocates one rank for a group, it delivers even lower throughput than the power-driven policy that uses more than one rank due to large footprint. The static method achieves the most power savings among all policies at the expense of noticeable performance degradation.

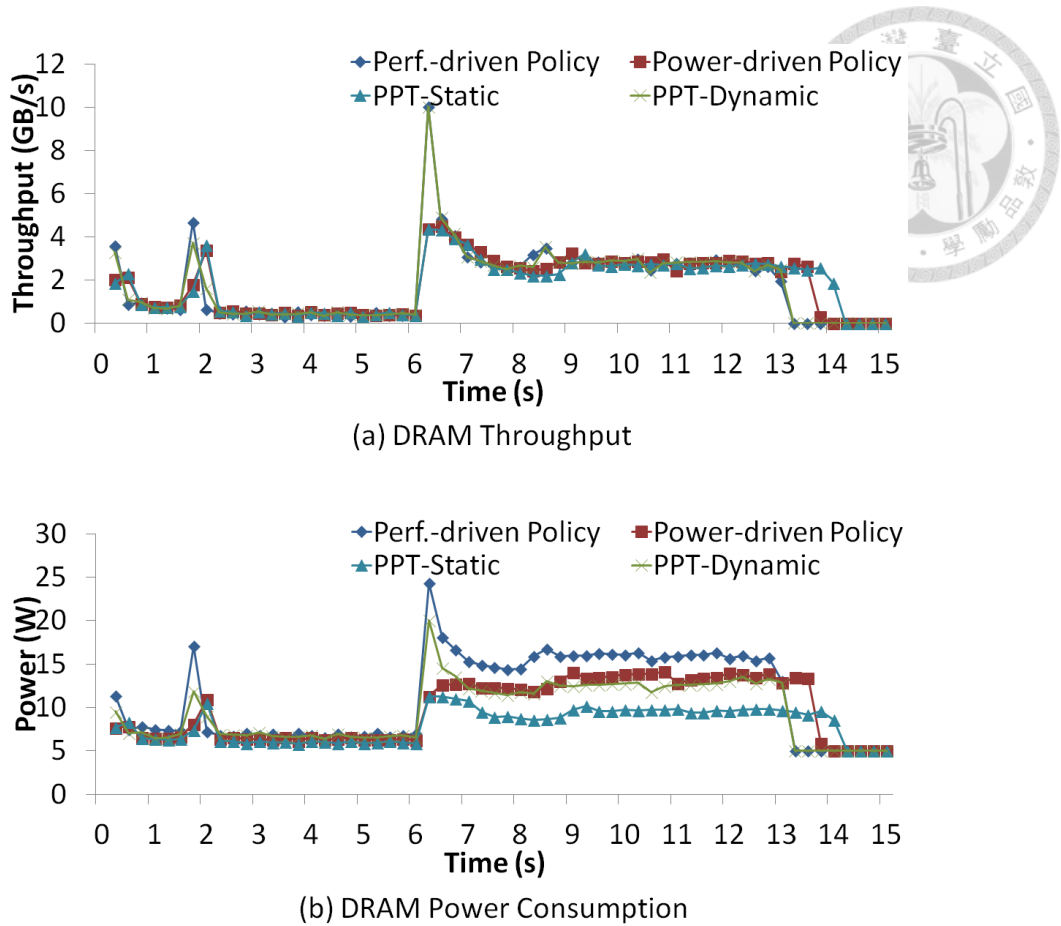


Figure 4.12: Throughput and power of DRAM system with SPECjbb for PPT evaluation.

4.3.2.2 Deferred Page Migration Analysis

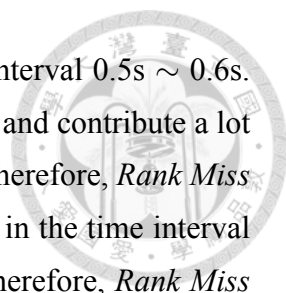
An important feature of *Adaptive Grouping* that requires further examination is the effectiveness of *Deferred Page Migration* triggered by new PPT configurations. *Deferred Page Migration* is designed assuming memory accesses presenting temporal locality. For SPEC2000-1, *Adaptive Grouping* with the dynamic BW estimation method changes the PPT configuration several times. Here, I show how access patterns conform to the new PPT configurations after the *Split* operation at 0.5s and the *Merge* operations at 4.6s and 4.9s in Figure 4.13(a) and (b), respectively. The sample time is 0.1s. I use the percentage of memory requests that do not hit the ranks of a group (*Rank Miss Rate*) as the metric for *Page Concentration*. The higher *Rank Miss Rate* means that memory behavior deviates more from the desired pattern of the new PPT configuration. The *Rank Miss Rate* is compared to *Concentration Threshold* (i.e., 0.1), and page migrations are triggered for *Page Concentration* when the *Rank Miss Rate* is higher than *Concentration Threshold*. On the



Figure 4.13: Rank Miss Rate, Hot/Cold Access Rate, New Page Rate and the number of page migrations for SPEC2000-1 with the dynamic BW estimation method in 0.5s~1.0s and 4.6s~5.1s for evaluations to Deferred Page Migration.

other hand, for *Page Distribution*, I use the ratio between the numbers of accesses of the hottest rank and that of the coldest rank (*Hot/Cold Access Rate*), to quantify if the memory accesses distribute to the designated ranks evenly. The ideal *Hot/cold Access Rate* is 1 (i.e., the numbers of accesses are equal among all designated ranks), and the higher *Hot/Cold Access Rate* indicates more deviated memory behavior from the new PPT configuration. When the *Hot/Cold Access Rate* is higher than *Distribution Threshold* (i.e., 2), page migrations are triggered for *Page Distribution*. I also show the *New Page Rate* (i.e., the ratio of new pages to total accessed pages in a sample period), and the number of page migrations.

In Figure 4.13(a), the *Rank Miss Rate* does not increase right after the *Split* op-



eration, because a lot of new pages are brought in during the time interval 0.5s ~ 0.6s. Due to temporal locality, the new pages are very likely to be hot data and contribute a lot of memory accesses which conform to the new PPT configuration. Therefore, *Rank Miss Rate* is low and only a few page migrations are triggered. However, in the time interval 0.8s ~ 0.9s, threads scheduled at that time access more old pages, therefore, *Rank Miss Rate* is higher and more page migrations are triggered. After 0.9s, *Rank Miss Rate* drops below 10%, and maintains stable thereafter. Figure 4.13(b) shows the similar behavior to Figure 4.13(a) except that page migrations are triggered for *Page Distribution* after *Merge* operations. As shown in Figure 4.13(b), during the time interval 4.6s ~ 4.7s, the *Hot/Cold Access Rate* does not increase right after the *Merge* operation at 4.6s, because there are many new pages. However, the *New Page Rate* decreases and the *Hot/Cold Access Rate* increases in 4.7s ~ 4.9s. The behavior after the *Merge* operation at 4.6s is quite similar to that after the *Split* operation at 0.5s. The other example in Figure 4.13(b) is the *Merge* operation at 4.9s, where the *Hot/Cold Access Rate* increases right after the *Merge* operation and it triggers more page migrations since fewer new pages are brought in.

The experiments show the efficiency of *Deferred Page Migration* which only moves the recently accessed MRU pages when the memory behavior does not conform to the new PPT configuration. I find that the moved pages are only 3% of total pages in the experiments. Moving a few pages is enough to make the access pattern conform to the new PPT configuration, because the MRU pages are very likely to be hot pages which contribute many memory accesses due to temporal locality. The access pattern always conforms to the new PPT configuration very quickly. After *Split* operations or the number of ranks per group decreases, the *Rank Miss Rate* always drops below 10% in 1s, and after *Merge* operations or the number of ranks per group increases, the *Hot/Cold Access Rate* always drops below 1.3 in 1s. The experimental results show that the number of memory requests incurred by page migrations is no more than 3% in the overall memory requests after PPT configuration changes. Therefore, the performance and power overhead are negligible.

4.3.2.3 Power State Analysis

The power reduction of the PPT framework comes from clustering memory requests into a subset of memory ranks and putting the remainings into low-power modes

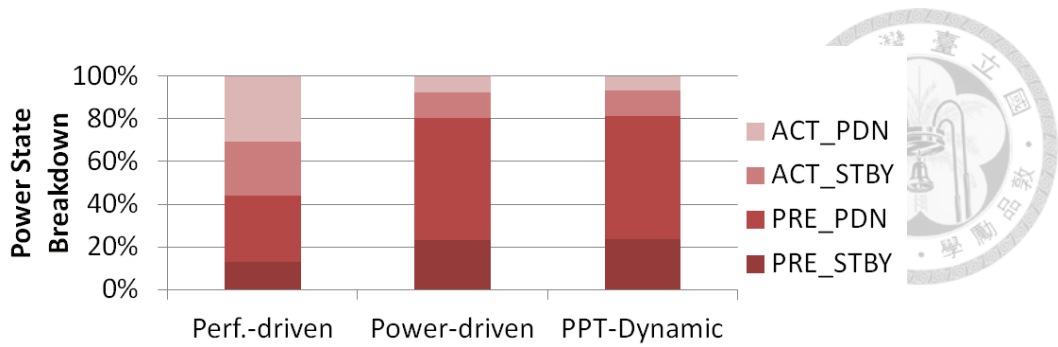


Figure 4.14: Power state breakdown of DRAM system with SPEC2000-1 for PPT evaluation.

to save energy. Therefore, it is important to know how much time the memory ranks are transferred into low-power modes, especially precharge power-down state, which is the state that consumes the least power. Figure 4.14 shows the power state breakdown of DRAM ranks with the performance-driven policy, the power-driven policy and the dynamic BW estimation method of *Adaptive Grouping* for SPEC2000-1. As shown in Figure 4.14, during the program execution, the memory system spends 31% of the time in the precharge power-down state with the performance-driven policy, but the memory system spends more than 57% of the time in the precharge power-down state with the power-driven policy and the dynamic BW estimation method to achieve significant power savings. This is because the memory requests are clustered into a subset of memory ranks. The idle ranks do not receive any memory request and do not perform row accesses. Therefore, there is no data in the row buffers of idle ranks after refresh operations and the row buffers can be turned off to save power.

4.3.2.4 Thermal Evaluation

The proposed PPT scheme controls memory temperature through *Group Switching* and *Activity Migration*. I first present the temperature profile of SPEC2000-1 to show how *Group Switching* performs. I then demonstrate the thermal effect of *Activity Migration* using SPEC2000-2. Please note that the temperature in the following discussion is the highest temperature among ranks at a time instance.

Figure 4.15(a) shows the temperature of the performance-driven, power-driven, and PPT mechanisms for SPEC2000-1. We could see that PPT maintains the lowest temperature among all schemes most of the time. The power-driven policy results in signifi-

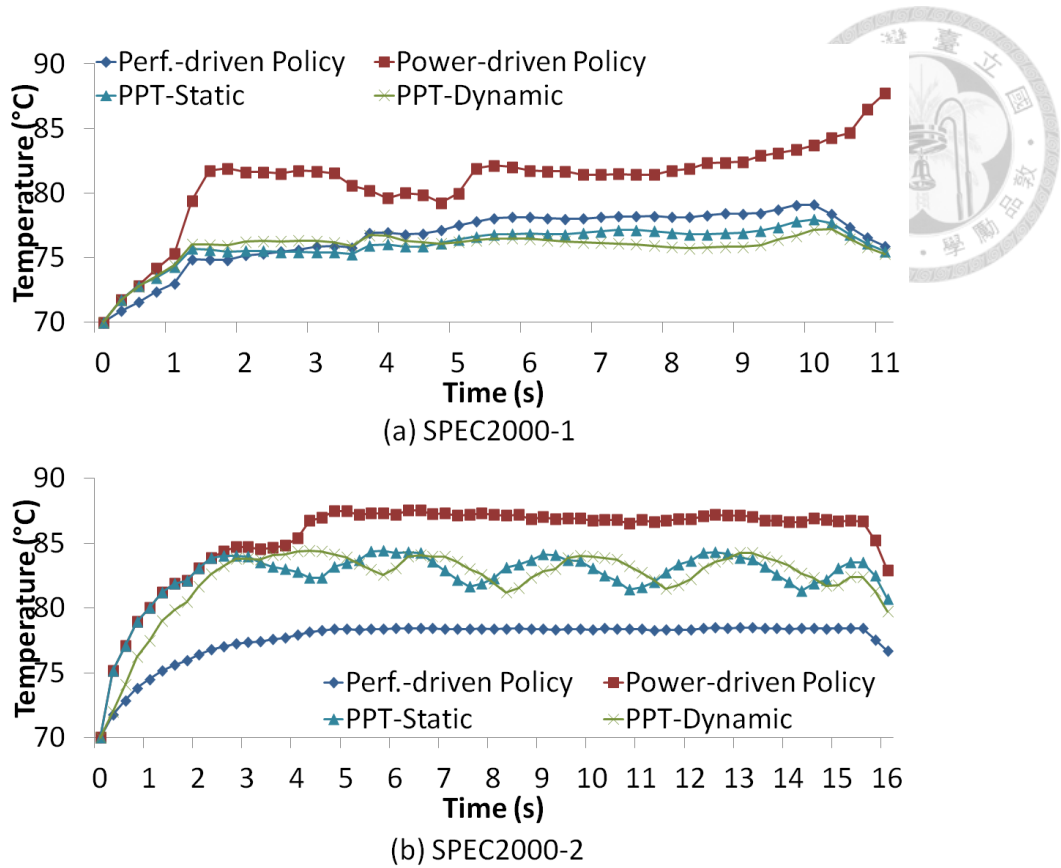


Figure 4.15: Temperature of DRAM system for PPT evaluation.

cantly higher temperature than PPT. For example, the power-driven policy achieves about 6.2°C higher peak temperature than the static BW estimation method at 2.75s and achieves about 6.9°C higher peak temperature than the dynamic BW estimation method at 9.25s. The peak temperature of the power-driven policy even exceeds 85°C. The performance-driven policy has pretty good thermal profile but it is still slightly higher than PPT. The temperature advantage of PPT over the performance-driven policy comes from lower power consumption of PPT as discussed in Section 4.3.2.1. Since the dynamic BW estimation method shows more power savings than the static BW estimation method, the temperature of the dynamic BW estimation method is also slightly lower than that of the static BW estimation method.

Figure 4.15(b) shows the temperature of the performance-driven, power-driven and PPT mechanisms for SPEC2000-2. Since the system has only one group for SPEC2000-2, memory temperature cannot be kept balanced through *Group Switching*. So both PPT mechanisms have higher temperature than the performance-driven policy, which distributes

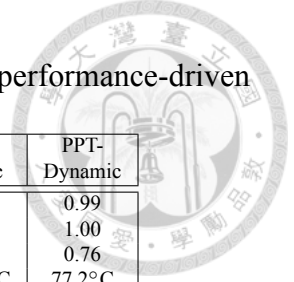


Table 4.3: Normalized throughput, power and peak temperature of the performance-driven policy, the power-driven policy and the PPT mechanisms.

		Performance-driven	Power-driven	PPT-Static	PPT-Dynamic
SPEC2000-1	Normalized Average Throughput	1.00	0.91	1.00	0.99
	Normalized Peak Throughput	1.00	0.49	1.00	1.00
	Normalized Power	1.00	0.74	0.83	0.76
	Peak Temperature	79.1°C	87.7°C	78.0°C	77.2°C
SPEC2000-2	Normalized Average Throughput	1.00	0.99	0.99	1.00
	Normalized Peak Throughput	1.00	0.95	0.95	1.00
	Normalized Power	1.00	0.58	0.67	0.70
	Peak Temperature	78.5°C	87.5°C	84.4°C	84.4°C
SPECjbb	Normalized Average Throughput	1.00	0.95	0.93	1.00
	Normalized Peak Throughput	1.00	0.45	0.43	1.00
	Normalized Power	1.00	0.85	0.68	0.84
	Peak Temperature	79.8°C	84.2°C	76.1°C	77.9°C

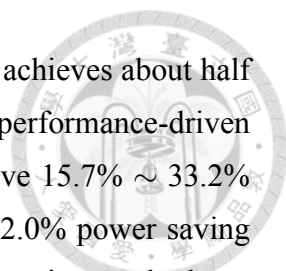
pages to all ranks. But with *Activity Migration*, I could control the temperature under a pre-defined threshold (i.e., 85°C in this case). In contrast, the power-driven policy has no way to control temperature, therefore, its peak temperature exceeds 85°C. The experimental results show that *Activity Migration* incurs about 7% more memory requests due to *Deferred Page Migration*. Therefore, the overheads of page migration is negligible. However, since the running threads may access two different ranks³, two ranks are active during *Activity Migration*. In this case, the power consumption of PPT is higher than that of the power-driven policy, which accesses only one rank, by about 10%.

4.3.2.5 Summaries of Performance, Power and Temperature of PPT

Table 4.3 shows the average throughput, peak throughput, power and peak temperature of the performance-driven policy, the power-driven policy, the static BW estimation method and the dynamic BW estimation method of *Adaptive Grouping* for SPEC2000-1, SPEC2000-2 and SPECjbb. Throughput and power are normalized to those of the performance-driven scheme.

We can see that both *Adaptive Grouping* ones can achieve comparable peak and average throughput to the performance-driven policy by utilizing sufficient channels and ranks to sustain bandwidth demand for SPEC2000-1 and SPEC2000-2. But the static BW estimation method suffers noticeable throughput degradation for SPECjbb, because it is not aware of the background I/O operations and underestimates the bandwidth de-

³The rank whose temperature exceeds the thermal threshold and the newly selected rank.



mand. For SPEC2000-1 and SPECjbb, the power-driven policy only achieves about half of the peak throughput of the dynamic BW estimation method and the performance-driven policy. For the power aspect, *Adaptive Grouping* mechanisms achieve 15.7% ~ 33.2% power savings, while the power-driven policy achieving 15.3% ~ 42.0% power saving compared to the performance-driven policy. The dynamic BW estimation method can achieve more power saving than the static BW estimation method for SPEC2000-1, because it can estimate the bandwidth demand more accurately to transit more ranks into the low-power mode for power reduction. But the static BW estimation method can achieve more power saving than the dynamic BW estimation method for SPEC2000-2 at the expense of performance degradation due to bandwidth underestimation in the initial phase. For temperature, both *Adaptive Grouping* mechanisms have the lowest peak temperature for SPEC2000-1 and SPECjbb, which have a lot of threads to partition threads into groups. For SPEC2000-2, the peak temperature of the power-driven policy exceeds 85°C, while both PPT mechanisms can keep memory devices from overheating. The peak temperature of the power-driven policy is 3 ~ 10°C higher than that of PPT. We can observe that, the dynamic BW estimation method is good in all cases among all policies to tradeoff performance, power and temperature.



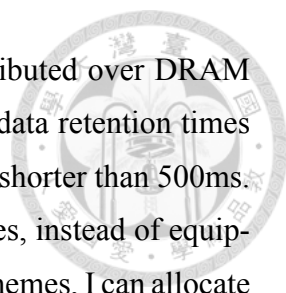
Chapter 5

SECRET Framework: DRAM Refresh Power Reduction

5.1 Introduction

As discussed in Chapter 1, one of the main challenges in low-power DRAM design is the inevitable refresh process. Researchers have proposed several approaches to reduce DRAM refresh power, such as disabling the refresh operations for memory blocks that have no data [83], or are recently recharged by memory accesses [77]. Liu et al. apply different refresh intervals to memory blocks independently according to their retention times [7]. Another approach is to prolong the refresh interval and adopt conventional Error Correcting Code (ECC) methods, e.g., Hamming code or Bose-Chaudhuri-Hocquenghem (BCH), to correct retention errors [97]. In these approaches, error correcting codes are applied to all DRAM cells. Therefore, it does not only come with significant area overheads, but also incur performance and energy penalties since decoding and encoding ECCs are required for every memory read/write. For example, in a conventional (72, 64) Hamming code, eight DRAM chips are paired with an extra chip, which requires 12.5% area overhead and additional power consumption. Wilkerson et al. [97] adopt the BCH code to reduce refresh power consumption of eDRAMs that are used as the last-level caches. In [97], all cache lines are protected by BCH, and a low-complexity decoding method can be adopted when there is no more than one error in a cache line. The area overhead of the method can be reduced by increasing the data size protected by BCH, but this incurs other overheads, such as bandwidth requirement. Although an optimization is proposed to solve this issue, the optimization is only suitable for caches, not for off-chip memories.

In this dissertation, I propose a novel error correction mechanism for retention errors in DRAMs, called SECRET (Selective Error Correction for Refresh Energy reduction). SECRET is developed based on the concept called Selective Error Correction.



Retention errors can be treated as hard errors, and are sparsely distributed over DRAM chips. According to Kim et al. [122], only $10^{-6}\%$ of the cells have data retention times shorter than 128ms, and $10^{-4}\%$ of the cells have data retention times shorter than 500ms. Therefore, if I have a priori knowledge of cells with high leakage rates, instead of equipping error correction capability in all memory cells as existing ECC schemes, I can allocate error-correcting bits for those leaky cells only and utilize a refresh interval that is longer than their data retention times to reduce refresh power. This observation leads to a very different error correction design for retention faults from prior works.

In this chapter, I describe the proposed SECRET framework in Section 5.2, and Section 5.3 shows the experiments for the SECRET framework.

5.2 The SECRET Framework

5.2.1 Main Idea

The SECRET framework is designed to prolong the refresh interval for reducing DRAM refresh power with a cost-effective error correction method that corrects the retention errors. Figure 5.1 shows the overview of the proposed SECRET framework. As mentioned in Chapter 1, the main design concept of SECRET is that with a priori knowledge of leaky cells under a refresh interval, I could construct a resource-efficient error correction scheme. Therefore, the center of SECRET is a *Selective Error Correction* (SEC) mechanism that equips the error correction capability only to identified leaky cells. To achieve more refresh power reduction (i.e., longer refresh interval), SEC must be able to tolerate more retention errors, which in turns require more resource overheads. Therefore, an architect must carefully evaluate the power savings versus overheads tradeoff to decide the target error rate that SEC is built for. A one-time profiling process to collect actual retention times of memory chips is performed before a machine is first used. Off-line profiling can be performed by system providers, or users if the profiling utility is built in the system. The addresses of leaky cells are stored in a file along with required error correcting bits. During system booting, the addresses of leaky cells and their correcting bits are then loaded into the main memory. To maintain the robustness against variations in data retention times due to the change in operating temperature, the SECRET framework adjusts the refresh interval at runtime to adapt to temperature variation. Below I describe

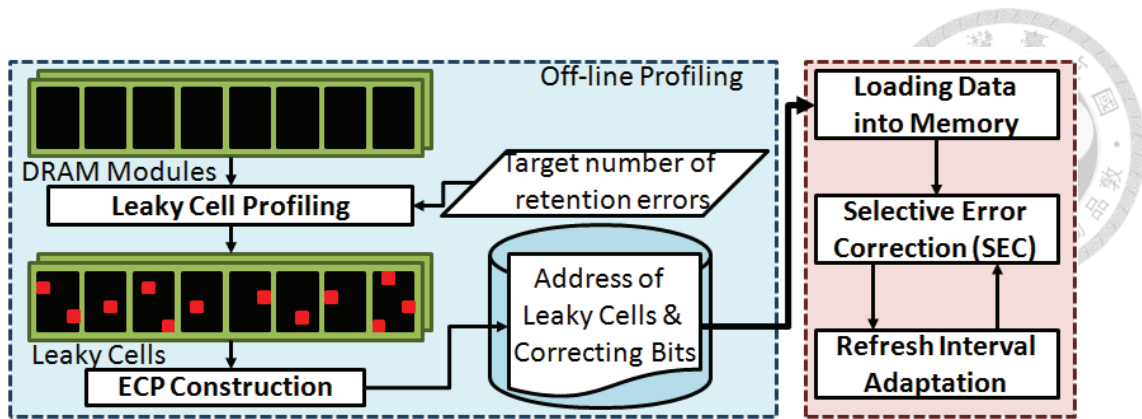


Figure 5.1: SECRET framework.

the details of the major building blocks of the proposed SECRET framework.

5.2.2 Candidates of the Error Correcting Scheme in SECRET

As discussed earlier, the retention errors must be corrected when increasing the refresh interval to reduce refresh power. In this section, I introduce three error correcting schemes that have been proved to be able to correct errors in main memories.

5.2.2.1 Hamming Code ECC

The Hamming code ECC is widely used to correct soft errors in DRAM modules. The Hamming code can correct one error in d data bits by p parity bits, where d is smaller than $2^p - p$ ($d < 2^p - p$). For example, for 64 data bits, at least 7 parity bits are required to correct 1 error. The Hamming code that is typically used in DRAMs has 8 parity bits to provide single error correction and double error detection capabilities for 64 data bits. This is also known as the conventional (72, 64) Hamming code, which is widely used in main memories due to its simple encoding/decoding process and short decoding time. To reduce refresh energy consumption, Emma et al. [78] proposed to increase the refresh interval and correct retention errors by the Hamming code. However, the disadvantage of the Hamming code is that it can only correct one error.

5.2.2.2 Bose-Chaudhuri-Hocquenghem (BCH) Code ECC

The BCH ECC is a polynomial ECC based on a finite field, and can correct more than one error. The BCH code can correct k errors in d data bits by using p parity bits,

where d is smaller than $2^{p/k} - p - 1$ ($d < 2^{p/k} - p - 1$). Although the BCH code can correct multiple errors and provide variable-strength error-correcting capability, the decoding and encoding processes of the BCH code are quite complex, and make it not suitable for time-critical main memories. Currently, the BCH code is widely used in secondary storage systems, such as NAND Flash. Zhang et al. [130] propose to use the BCH code to correct multiple wear-out errors in the phase change memory that is utilized as a part of the main memory system. Targeting at platforms utilizing eDRAMs as the last-level caches, Wilkerson et al. [97] use BCH code to reduce the refresh power consumption of eDRAMs.

5.2.2.3 Error Correcting Pointer

Error Correcting Pointer (ECP) [1] is proposed for correcting wear-out errors in phase change memories. Since wear-out errors can be known in advance, only faulty cells are equipped with ECPs. The structure of an ECP is shown in Figure 5.2(a). An ECP entry is composed of p -bit correction pointer and a replacement cell. Among the 2^p data bits, the correction pointer indicates which bit has a hard error. The replacement cell stores the correct data. For example, in Figure 5.2(a), the 510-th bit is known to be a faulty cell. Its ECP has the pointer set to 510, and the replacement cell stores the correct data. A p -to- 2^p row decoder is required to align the replacement cell with the error to perform decoding. Figure 5.2(b) shows the logic of ECP_1 decoder [1] that corrects one error in 512 data bits. The ECP scheme is optimized for wear-out errors that are stuck at a fixed value by using a differential encoding where the replacement bit conditionally inverts the failed bit. ECP is very flexible to correct multiple errors since each identified error is protected by a specific ECP. In the meantime, the ECP scheme keeps its decoding simplicity for correcting multiple errors by correcting errors one-by-one. However, the ECP scheme can only handle hard errors that are known in advance, but not soft errors since ECPs have no ability of detecting errors.

5.2.3 Selective Error Correction (SEC)

To design the SEC mechanism, there are two main design issues. First, what kind of error correction method should be used to correct retention errors? Second, how do I locate memory cells with retention errors at runtime?

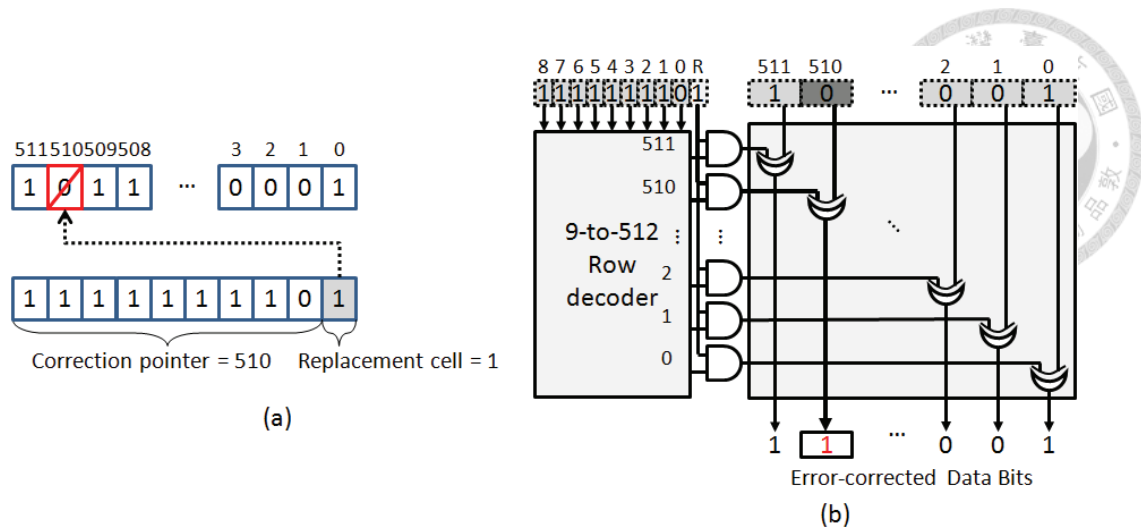


Figure 5.2: (a) Error correcting pointer. (b) Hardware implementation for ECP_1 decoder [1].

A good error correction method for SEC should support variable-strength error-correcting capability. The distribution of retention errors is random within a DRAM chip. That is, the number of retention errors of a memory block protected by error-correcting bits varies. Since retention errors can be treated as hard errors, all the leaky memory cells that are identified at off-line need to associate with error-correcting bits. Therefore, the most cost-effective error correction method for SEC is to allow memory blocks with more retention errors to have stronger error correction capability, and vice versa. To achieve this, the ECP scheme mentioned in Section 5.2.2.3 is a perfect candidate for retention errors¹. The number of ECPs of a protected memory block is equal to the number of faulty cells. Although BCH is also able to provide variable-strength error-correcting capability, its decoding and encoding are much more complex than that of ECP.

To locate memory cells with retention errors at runtime, I partition memory space into equal-sized regions. As shown in Figure 5.3, ECPs of the same memory region are placed contiguously in the memory, and the ECP directory is used to index the ECPs. The ECP directory is indexed by the region number. Each entry of the ECP directory records the number of ECPs in the corresponding memory region and the physical address of the first ECP of the region. For each memory request, the ECP directory is checked to see

¹Bit-fix [106] also provides the same capability. I choose the ECP scheme, but bit-fix can also be adopted in SECRET.

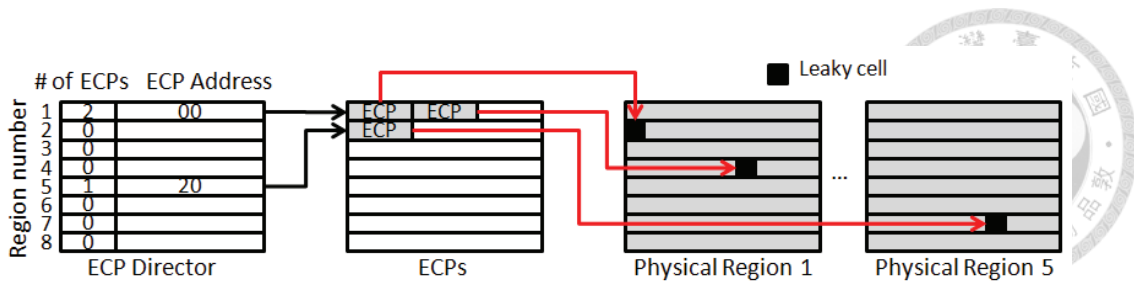


Figure 5.3: Error correcting information for the SEC mechanism.

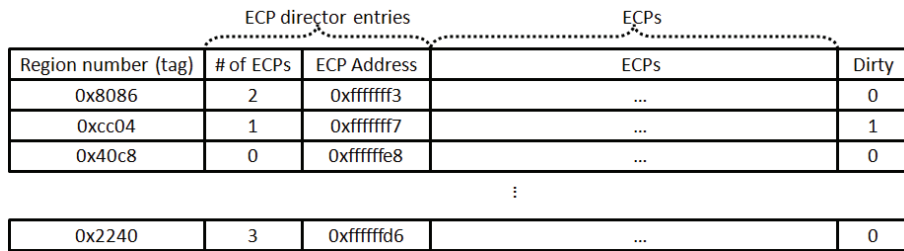
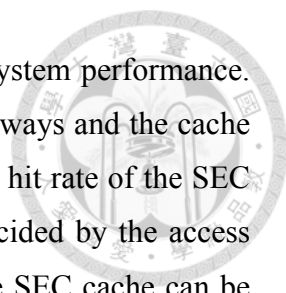


Figure 5.4: Architecture of the SEC cache.

if the corresponding memory region contains faulty cells (i.e., a non-zero number in the field of the number of ECPs). If there are faulty cells, all ECPs of the memory region are fetched from the memory, and the memory controller compares the pointers of ECPs and the address of the requested data. The ECPs that match the address are decoded to perform error correction.

From the above discussion, we can see that the SEC mechanism introduces extra memory requests for fetching the ECP directory and ECPs. To minimize the performance overhead of additional memory accesses, I propose an optimization to cache the ECP directory and ECPs in the memory controller. When the ECP directory entry and ECPs of a region are fetched from memory to the memory controller, they are kept in the SEC cache in the memory controller. Figure 5.4 shows the architecture of the SEC cache. Each SEC cache line stores the information of one ECP directory entry and all the ECPs in that region. To have an ECP directory entry and ECPs of a memory region cached in the same set, the SEC cache is indexed by region numbers instead of the physical addresses of the ECP directory or ECPs. The dirty bit is used to indicate if the replacement cells of ECPs are updated. If a replacement is required, a write-back operation of ECPs is triggered if the corresponding dirty bit is set. Please note that the ECP directory part in the SEC cache is read-only and does not have to be written back to memory.



The decision of the SEC cache configuration is crucial for system performance. The cache configuration includes the number of sets, the number of ways and the cache line size. The numbers of sets and ways of the SEC cache affect the hit rate of the SEC cache. Since a design that achieves acceptable cache hit rate is decided by the access pattern of the memory regions, the numbers of ways and sets of the SEC cache can be decided along with the configuration of the micro-architecture during system design time, which performs design space exploration with the target workloads.

For the SEC cache line size, it decides the number of ECPs in a region that can be cached. So, to decide the appropriate cache line size, an architect needs to estimate the number of retention errors in a region given an error rate. Here I assume the geometric distribution of retention times is uniformly random distribution. Since the leakage currents that cause retention errors are from several different sources, e.g., band-to-band tunneling current, body effect, etc. [131], which may have different geometric distribution properties. So, it is reasonable to assume that retention errors are uniformly randomly distributed among DRAM chips. Moreover, memory cells in a region do not usually locate in the same chip. In a typical DRAM system design, the requested data are interleaved in all DRAM chips in a rank. For example, in an 8-chip DRAM rank, a contiguous 64-byte data block is partitioned into eight 8-byte data blocks, which are distributed across eight DRAM chips in this rank. Therefore, I believe that strong locality among retention errors are not common cases. Extreme cases only happen when excursions occur during DRAM manufacturing. However, to be able to accommodate variations among real DRAM chips, I could set the line size larger than what is estimated with data generated based on uniformly random distribution.

Figure 5.5 shows the system architecture of the SEC mechanism. The memory controller contains the SEC cache and the *Error Correcting Unit*. The *Error Correcting Unit* is similar to the ECP_1 decoder described in [1] that decodes one ECP at one time. However, unlike phase change memories, the data values of faulty DRAM cells with retention errors are not stuck at 0 or 1, and the differential encoding is not applicable. I explicitly store the replacement value in the replacement cell and slightly modify the decoder accordingly. When the memory controller receives a memory request, the memory controller issues requests to both the memory arrays and the SEC cache. If the requested

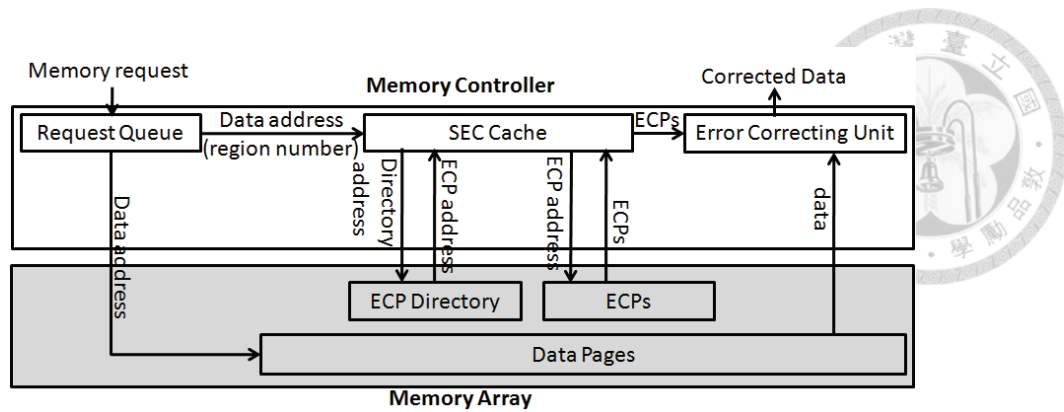


Figure 5.5: System architecture of the SEC mechanism.

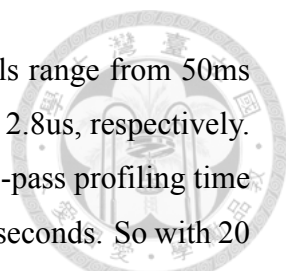
ECPs are found in the SEC cache, the error correction information is provided to the *Error Correcting Unit* from the SEC cache directly. In this case, no extra memory requests are issued. If it is an SEC cache miss, the memory controller first issues a memory access to fetch the ECP directory entry from memory. One or more memory accesses for the ECPs are needed only when the ECP directory entry indicates that there are errors in the requested region.

5.2.4 Off-line Phase: ECP Directory/ECPs Construction

During the off-line phase, I first identify memory cells with retention errors given a target error rate and then build the ECP directory and ECPs accordingly.

I characterize data retention times of DRAM cells in a method similar to the testing process proposed in [81] and [92]. The process prolongs the refresh interval incrementally and checks if the DRAM cells can retain data. To test the DRAM chips, all refresh operations are disabled and the row buffers are managed by the close-page policy. The testing process writes all 1's to the chips, waits for a refresh interval, and reads the data back to see if they are intact. The process is executed until the number of retention errors meets the target number, or I find that the number of errors in a region meets the number of ECPs that an SEC cache line can keep.

The off-line profiling process is performed once before a machine is first used, and the time required for retention time testing is quite small. As mentioned above, the profiling process includes (1) writing data into the DRAM, (2) waiting for the target retention time, and (3) reading the data out. Assume that I perform the profiling process on

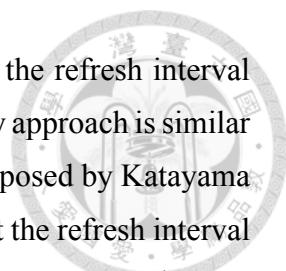


a 4GB DRAM, which has 512K rows, and the tested refresh intervals range from 50ms to 1s. According to [132], writing and reading a row needs 3.4us and 2.8us, respectively. Since Step (2) can be overlapped with reads/writes of other rows, one-pass profiling time can be estimated as reading/writing 512K rows, which is equal to 3.3 seconds. So with 20 refresh threshold steps, the overall profiling process takes about 66 seconds. If the DRAM system has multiple memory controllers and/or multiple memory channels that allow the DRAM rows to be read or written simultaneously, the testing of retention times can be performed in parallel to reduce the time required for the off-line profiling process.

With the knowledge of the physical positions of leaky cells, I then construct the ECP directory and ECPs. Both the ECP directory and ECPs need to be placed in non-leaky cells. Since the whole ECP directory needs to be placed in contiguous memory cells, it is possible that I am not able to find a big enough memory segment without any retention errors. In this case, I adopt the Triple-Modular Redundancy (3-MR) ECC that duplicates the original data twice to protect the ECP directory. To avoid three separate accesses to read an ECP directory entry that is protected by 3-MR, I place the three copies of an ECP directory entry (108 bits = 14 bytes) consecutively in the same 64B data block so that all three copies can be accessed in one memory access. A majority vote is performed at the memory controller when these three copies of the ECP directory are different. The logic for the majority vote for each bit only requires 3 AND gates and 2 OR gates, so the logic is quite simple and very fast. The only situation of having uncorrectable errors with 3-MR is that the same bit positions of two copies of an ECP directory entry have retention errors. The probability for this to happen is extremely low. For example, in a 4GB DRAM system with region size of 128KB and 10^{-6} retention error rate, the probability is 3.5×10^{-6} . The placement of ECPs is more flexible than that of the ECP directory. Only the ECPs of the same region need to be stored contiguously in memory not the whole ECPs. The constructed ECP directory (along with its starting address) and ECPs are stored in a file. During system booting, these error correction information are then loaded into memory.

5.2.5 Refresh Interval Adaptation

The data retention capability of memory cells are subjected to various system perturbations, like temperature. Higher temperature leads to larger leakage current. There-



fore, to ensure the robustness of the proposed SECRET framework, the refresh interval needs to be adjusted dynamically to adapt to temperature variation. My approach is similar to the runtime refresh interval adaptation method in the prototype proposed by Katayama et al. [92]. The basic idea of the refresh interval adaptation is to adjust the refresh interval so that the number of retention errors are kept constant. That is, when unexpected retention errors are detected, the refresh interval should be scaled down. On the other hand, when expected retention errors are not detected, the refresh interval should be scaled up. To support this, in addition to identifying the memory cells with retention errors given a target error rate, I also need to identify the memory cells that may cause retention errors when fluctuations of temperature occur, and monitor these bits periodically. So, in Section 5.2.5.1, the process of identifying the memory cells for refresh interval adaptation is described. Moreover, I also need to identify the maximum leakage variation during the monitoring period so that I can guarantee that all memory cells that may cause retention errors due to temperature change are identified. The calculation of the maximum leakage variation ratio is described in Section 5.2.5.2. Then, I show the proof of the correctness of the refresh interval adaptation in Section 5.2.5.3.

5.2.5.1 Profiling Memory Cells for Refresh Interval Adaptation

As shown in Figure 5.6, ref_n indicates the refresh interval value that reaches the target error rate. The set of bits with retention times between ref_n and ref_{n+1} are the DRAM cells that have the shortest data retention times among the DRAM cells that are not expected to have retention errors if system temperature is close to the setting of the off-line profiling. This set of bits are indicated as Set 2 in Figure 5.6. Memory cells in Set 2 should also be protected by ECPs to prevent retention errors when the leaky current increases. On the other hand, as shown in Figure 5.6, Set 1 is the set of DRAM cells that have the longest data retention times among those DRAM cells that may have retention errors. So, in the SECRET framework, bits in Set 1 and Set 2 are periodically checked and corrected. When there are no retention errors in Set 1, this indicates that the length of the refresh interval can be increased to further reduce refresh power since the number of retention errors is lower than expected. When there are retention errors in Set 2, the length of the refresh interval should be reduced to guarantee the correctness of the DRAM system.

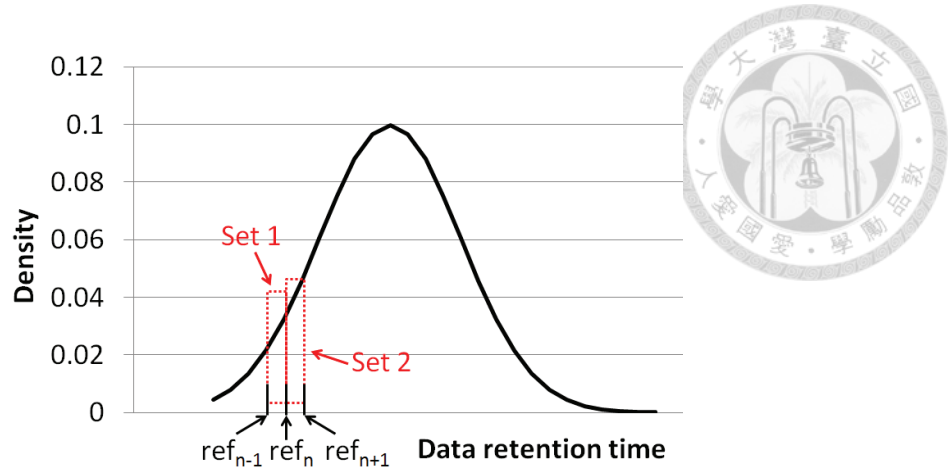


Figure 5.6: Data retention time distribution of DRAM cells and bits in Set 1 and Set 2.

5.2.5.2 Deduction of the Worst Case Leakage Ratio

To guarantee that all retention errors are protected by ECPs, I need to ensure that Set 2 covers all the bits that may cause retention errors due to temperature change under the worst case during the monitoring period. To achieve this, I need to find out the maximum leakage variation ratio, i.e., maximum/minimum leakage current, during the monitoring period under the extreme temperature change condition. Since retention time variation is proportional to that of leakage current, the maximum leakage variation ratio is also the maximum retention time variation ratio during the monitoring period. Let t denotes the monitoring period, and R denotes the maximum retention time variation (leakage variation) when chip temperature is T in monitoring period t . I can deduce that, to guarantee Set 2 (Set 1) to cover all the bits that should be checked periodically with the maximum leakage variations within the monitoring period, ref_{n+1} (ref_{n-1}) in Figure 5.6 should be set as $ref_n \times R$ (ref_n/R).

As mentioned earlier, the maximum retention time variation R is proportional to the maximum variation of leakage current during the monitoring period t . According to [133], when the chip temperature is T , the amount of leakage current $I_0(T)$ is modeled by the following equation.

$$I_0(T) = I_0(25^\circ C) \times (T/300)^{1.8}. \quad (5.1)$$

Therefore, when the chip temperature is T , and the maximum temperature increase in t is ΔT , the maximum leakage variation ratio $R(T, t)$, which is also the maximum retention

time variation ratio of the period t , can be calculated by the following equation.

$$R(T, t) = \frac{I_0(25^\circ C) \times ((T + \Delta T)/300)^{1.8}}{I_0(25^\circ C) \times (T/300)^{1.8}} = ((T + \Delta T)/T)^{1.8}. \quad (5.2)$$

According to [44], given the chip temperature T and monitoring period t , ΔT can be modeled by

$$\Delta T = ((TR \times P + T_{amb}) - T) \times (1 - e^{-t/\tau}), \quad (5.3)$$

where TR is the thermal resistance, P is the peak power consumption of the chip, T_{amb} is the ambient temperature, and τ is the time for the temperature difference between T and $(TR \times P + T_{amb})$ to be reduced by $1/e$.

So, assuming the chip has maximum and minimum working temperature of T_{max} and T_{min} , the maximum to minimum retention time ratio R in the extreme case is

$$R = \max(R(T, t)), T_{min} \leq T \leq T_{max}. \quad (5.4)$$

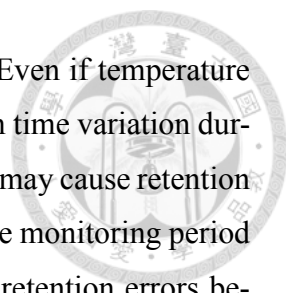
Therefore, as mentioned earlier, Set 2 (Set 1) should cover all the bits with retention times between ref_n and $ref_n \times R$ (ref_n and ref_n/R). Equipping ECPs for bits in Set 2 can guarantee that there are no uncorrectable retention errors even with extreme temperature change during the monitoring period. I will give the proof of the correctness for the refresh interval adaptation process in the next subsection.

5.2.5.3 Proof of the Correctness of Refresh Interval Adaptation

According to Section 5.2.5.2, the bits without any error correcting capabilities are those with retention times that are longer than $ref_n \times R$. I can prove that, there are no uncorrectable errors during the period t when refresh interval adaptation process is performed. That is, no uncorrectable retention errors occur no matter the refresh interval is prolonged or shortened.

Now, I show that bits with retention times longer than that of the bits in Set 2 are guaranteed not to have retention errors in any cases. The proofs of all cases are as follows:

Maintaining the refresh interval : Assume the current refresh interval is ref . If I find errors in Set 1, but not in Set 2, this indicates that there is no obvious change in



chip temperature, and the refresh interval should be still ref . Even if temperature changes after adaptation, since I can guarantee that the retention time variation during time interval t in the worst case is R , at most the bits in Set 2 may cause retention errors due to temperature change under the worst case during the monitoring period t . Therefore, non-protected bits are guaranteed to not having retention errors because these bits have retention times longer than those bits in Set 2 in the worst case.

Prolonging the refresh interval : Assume the current refresh interval is ref . If I find no errors in Set 1, this indicates that the chip temperature drops, and the refresh interval can be prolonged from ref to $ref \times R$. If temperature does not change, at most the bits in Set 1 may cause retention errors due to refresh interval adaptation. Even if temperature changes after adaptation, since I can guarantee that the retention time variation ratio during time interval t in the worst case is R , at most the bits in Set 2 may cause retention errors due to temperature change under the worst case during the monitoring period t . Therefore, non-protected bits are guaranteed to not having retention errors because these bits have retention times longer than those bits in Set 2 in the worst case.

Shortening the refresh interval : Assume the current refresh interval is ref . If I find errors in Set 2, this indicates that the chip temperature raises, and the refresh interval should be shortened from ref to ref/R . Even if temperature raises again after adaptation, since I can guarantee that the retention times are also shortened by R times in the worst case during time interval t , at most the bits in Set 2 may still cause retention errors due to temperature change under the worst case during the monitoring period t . Therefore, non-protected bits are guaranteed to not having retention errors because these bits have retention times longer than those bits in Set 2 in the worst case.

Therefore, non-protected bits are guaranteed to not have retention errors because these bits have retention times longer than those bits in Set 2. As a result, the correctness of the adaptation process can be guaranteed in all cases.



Table 5.1: Number of bits/bytes of each ECP directory/ECP field in the SECRET framework.

An ECP Directory Entry	
Num. of ECPs	4 bits \times 3 = 12 bits
ECP address	32 bits \times 3 = 96 bits
Total	14 bytes
ECP	
Correction pointer	20 bits
Replacement cell	1 bit
Total	3 bytes

5.2.6 Discussion on Overheads of SEC

ECP Directory/ECPs: The number of bits/bytes required by each field in the ECP directory and ECPs are listed in Table 5.1. As mentioned in Section 5.2.4, I use the 3-MR technique to protect the ECP directory. So, I have three copies of each ECP directory entry which needs 108 bits (14 bytes) in total. The memory space allocated to the ECP directory is related to the number of regions only and independent of the number of tolerable retention errors. For ECPs, because one ECP is used to correct one retention error, the amount of memory space occupied by ECPs is related to the number of retention errors. Assume I have a 4GB DRAM system, which is partitioned into 32K regions where each of the region is 128KB, each ECP needs 20 bits for recoding the position of the error in the region, and 1 bit for the replacement cell. With 10^{-7} retention error rate, the ECP directory and ECPs occupy only 0.01% of the memory space. Even with the retention error rate up to 10^{-4} , the memory space required for the ECP directory and ECPs is still very small, only 0.24%.

Error Correcting Unit: My *Error Correcting Unit* adopts the ECP_1 decoder similar to the one shown in Figure 5.2(b). The ECP_1 decoder has a 9-to-512 row-decoder to align the replacement cell with the leaky cell [1]. The ECP_1 decoder has a reasonably small latency which is no more than one processor cycle [1], but only one ECP can be decoded at a time. The data block size of a memory access is processed in the *Error Correcting Unit* at a time. In a DDR3 memory, the block size is 64 bytes. The possibility of having multiple errors in such a small unit is quite low. In my experiments, 0.05% of data blocks have one error, and only 1.04×10^{-5} % of data blocks have two errors. No data blocks have three or more errors. Therefore, one or two ECPs are decoded only in very few cases. For the area overhead, my synthesis results show that the *Error Correcting*

Unit needs only 0.014mm^2 area with 45nm technology, which is negligible compared to the memory controller.

SEC Cache: The SEC cache size is determined by three configuration parameters, associativity, the number of sets, and cache line size. The cache line size affects how many retention errors SEC can tolerate in a region, while the associativity and number of sets decide the hit rate of an SEC cache, which is workload dependent, not DRAM size dependent. A larger SEC cache can reduce extra memory accesses and tolerate more errors at the expense of higher overheads of accessing the SEC cache itself. As mentioned in Section 5.2.3, the design of SEC cache is workload related and should be performed along with the micro-architecture during the system design time. In Section 5.3.2.1, I will present a systematic way to make a right design decision for the SEC cache.

Refresh Interval Adaptation: The monitoring overheads of the refresh interval adaptation process mainly come from the extra memory accesses for reading the bits in Set 1 and Set 2 along with their ECP directory and ECPs. Assuming the monitoring performed for every second and the operating temperature between 25°C and 85°C , I can infer that the maximum leakage variation R is equal to 1.003607 according to the method discussed in Section 5.2.5. So if I have a 4GB DDR3-1333 SDRAM system partitioned into 32K regions, with the retention time distribution taken from [122] and 10^{-6} target error rate, Set 1 and Set 2 contain 694 and 719 bits, respectively. Therefore, reading Set 1 and Set 2 incurs about 91KB/s bandwidth overheads, and accessing the ECP directory and ECPs incurs about 4MB/s bandwidth overheads. Since the channel bandwidth of a DDR3-1333 SDRAM is 10.66GB/s, the bandwidth overheads incurred by the adaptation process take no more than 0.1% of DRAM channel bandwidth.

5.3 Evaluation to SECRET Framework

5.3.1 Experimental Setup

The simulation framework used in this chapter is composed of three components: Wind River SIMICs [134], Ruby of Gems [135] and DRAMsim [124]. SIMICs is a full system simulator that can execute target benchmarks on unmodified operating systems. To simulate memory and cache in details, Ruby that is integrated with DRAMsim is loaded into SIMICs. I simulate a Sun virtual machine called Abisko that runs a version



Table 5.2: System configurations for SECRET evaluation.

Parameters	Value
Processor L1 caches (per core) L2 cache (shared)	4-core, 2GHz, 1 hardware context per core 64KB Inst/64KB Data, 2-way, 64B line 8MB, 16-way, 64B line
DRAM Memory controller Memory rank Memory bank Channel bandwidth Row buffer management Refresh configuration	4GB, 2 channels, 2 DIMMs/channel, 1 rank/DIMM 32-entry request buffer 1GB, 1333MHz DDR3-SDRAM, 8 banks/rank 16384 rows/bank, 1024 columns/bank 8-byte/channel, 10.66GB/s open-page policy 64ms, one-channel, one-rank, all-bank policy
The SECRET framework ECP directory	region size of 128KB, 32K regions in a 4GB DRAM 32K entries

Table 5.3: Workloads for SECRET evaluation.

Workload suit	Details
SpecJBB	4 warehouses
Spec2006	
Mix1	401.bzip2, 464.h264ref, 453.povray and 447.dealII
Mix2	410.bwaves, 456.hmmer, 400.perlbenc and 471.omnetpp
Mix3	454.calculix, 416.gamess, 435.gromacs and 450.soplex
Mix4	482.sphinx3, 444.namd, 434.zeusmp and 437.leslie3d
Mix5	470.lbm, 445.gobmk, 473.astar and 403.gcc
Mix6	462.libquantum, 429.mcf, 433.mile and 458.sjeng
PARSEC	blackscholes, bodytrack, ferret, fluidanimate streamcluster, swaptions, vips, x264, canneal

of Solaris 10. All benchmarks are executed on a 4-core CMP system sharing an 8MB L2 cache with 4GB DDR3-SDRAM, whose power parameters are set according to Micron MT41J128M8JP 1Gb DDR3-SDRAM [132]. I implement the power management policies of DDR3 systems, which have fast-exit and slow-exit modes when entering the idle state. The detailed system configurations are listed in Table 5.2. Besides the baseline 4-core system with an 8MB L2 cache, I also evaluate the SECRET framework with a 4MB L2 cache and a 2MB L2 cache that may be used in a relatively low-end system to see the effectiveness of SECRET with a small cache and relatively high memory bandwidth demand. For SECRET, the size of a region is set to 128KB, and the DRAM is partitioned into 32K regions. In the SECRET framework, the row buffer is closed as frequently as the baseline configuration, which is the DRAM system without the SECRET framework and uses 64ms refresh interval. I evaluate SECRET on three categories of workloads: SPECjbb for on-line transaction processing, PARSEC for multi-threaded applications, and mixtures of SPEC CPU2006 for multi-programming workloads. Table 5.3 lists the details of each workload. Each set of benchmarks is simulated for 1 billion cycles after fast-forwarding the first 0.5 billion cycles.

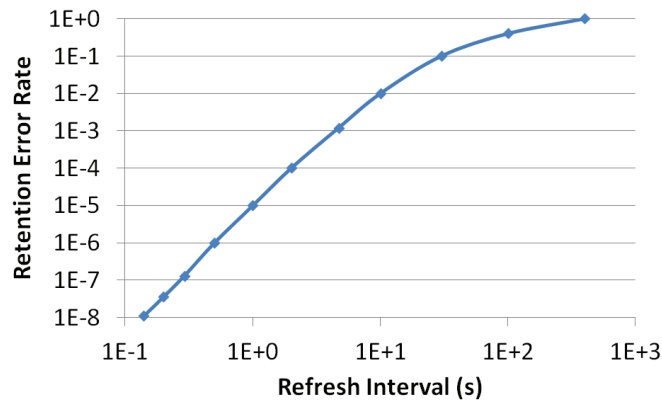


Figure 5.7: Retention error rates of utilizing various refresh intervals.

For the experiments presented in this chapter, the retention time distribution is taken from [122], which presents measurements from real DRAM chips. Figure 5.7 shows the relation between error rates and refresh intervals deduced from the relation between cumulative failure probability and retention time shown in [122]. As mentioned in Section 5.2.3, I assume the leaky cells of a DRAM chip are uniformly and randomly distributed in the chip. According to this assumption, I generate five different geometric distributions of leaky cells for evaluation. In the five distributions, with 10^{-6} retention error rate, the average number of retention errors of a region with 128KB size for all five distributions is one error. The maximum number of retention errors in a region of the five distributions are seven, seven, eight, eight and nine, respectively. However, I also have interest in how the SECRET framework works when the distributions of the leaky cells in DRAM chips have some spatial locality. So, I also generate other 60 distributions that have several different degrees in spatial locality of leaky cells. In these 60 distributions, with 10^{-6} retention error rate, the average number of retention errors of a region with 128KB size is still one error, but the maximum numbers of retention errors in a region are ranging from seven to thirteen. I will introduce these 60 distributions in Section 5.3.2.5 in detail.

5.3.2 Experimental Results

In this section, I first demonstrate how to decide the target error rate and the SEC cache configuration. For this set of experiments, I use only one of the leaky cell distribu-

tions for demonstration, which has a maximum of eight retention errors in a region with 10^{-6} retention error rate. I then discuss the energy and performance behavior of SECRET on the five generated distributions. After that, I discuss the effectiveness of the SECRET framework with reduced last-level caches and leaky cell distributions with spatial locality, and then compare the SECRET framework with the conventional Hamming Code ECC for refresh power reduction.

5.3.2.1 Design Space Exploration: Deciding Target Error Rate and SEC Cache Configuration

The target error rate affects both refresh power reduction and error correction overheads. With higher target error rates, I can increase the refresh interval, which is inversely proportional to refresh power consumption. Therefore, the refresh power reduction can be correlated with refresh intervals using the following formula:

$$RP_{T_{ref}} = 1 - (T_{ref_min}/T_{ref}), \quad (5.5)$$

where $RP_{T_{ref}}$ represents the percentage of refresh power reduction achieved by the operating refresh interval T_{ref} , and T_{ref_min} represents the worst-case refresh interval. Therefore, based on Figure 5.7, I can derive the relation between refresh power reduction and error rates as shown in Figure 5.8. Since refresh power reduction saturates when the retention error rate is larger than 10^{-4} , I only have to consider the retention error rate that is smaller than 10^{-4} . Target error rates also affect SEC overheads since an SEC cache line needs to store all the ECPs in a region as discussed in Section 5.2.6.

To estimate the SEC cache overheads of various cache line sizes, I first need to determine its associativity and number of sets, which affect the hit rate of the SEC cache. With higher hit rates, I can minimize extra memory accesses to fetch ECPs/ECP directories. My experiments indicate that the effect of 1% SEC cache miss rate on the overall performance is negligible. The average SEC cache miss rates of all tested workloads with different number of sets and ways are shown in Figure 5.9. I observe that only the 256-set/4-way, 512-set/4-way, 1024-set/2-way, and 1024-set/4-way cache configurations can achieve miss rate that is below 1%. Therefore, I select 256-set/4-way as my SEC cache configuration.

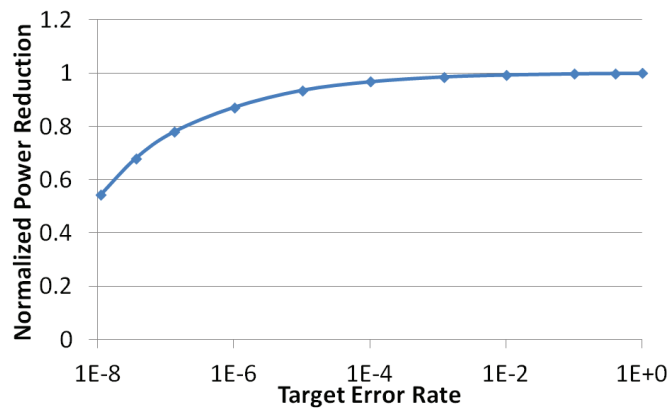


Figure 5.8: Refresh power reduction achieved by utilizing refresh intervals of various target retention error rates.

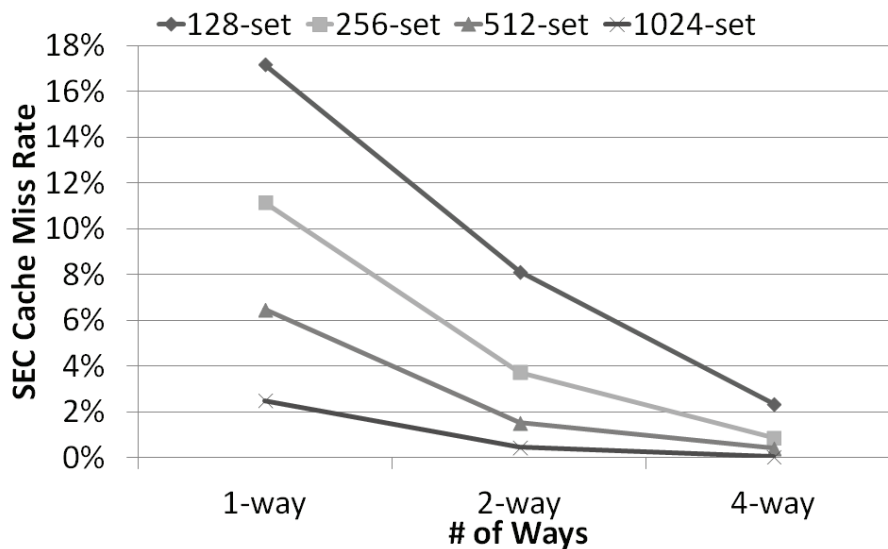


Figure 5.9: Average cache miss rate of SEC cache with varied number of ways and sets.

With the associativity and set numbers decided, I now examine how target error rates affect the SEC cache line size. Figure 5.10 shows the DRAM power reductions when various cache line sizes of the 256-set/4-way SEC cache are utilized. The results are normalized to the power consumption of the baseline DRAM, and each of the point is the average of all tested workloads. Refresh power reduction for a given error rate is derived from Figure 5.8. I can observe that going beyond 10^{-6} error rate, it only brings little improvement in refresh power reduction while the SEC cache line size increases steadily. Therefore, I choose 10^{-6} as the target retention error rate, which has 500ms



Figure 5.10: SEC cache line size vs. DRAM power reduction under various retention error rates.

refresh interval according to Figure 5.7. Recall that I leave margins to tolerate distribution variation among DRAM chips by setting the line size larger than what is estimated as discussed in Section 5.2.3. Here I set the cache line size to 1.5 times of the estimated size. Since the maximum number of retention errors of a region in the leaky cell distribution studied in the design space exploration is eight with 10^{-6} error rate, the SEC cache line size is set to 36 bytes, which is able to store twelve ECPs and the ECP directory entry of a region. In my system setup, when the target error rate is set to 10^{-6} , the probability of having more than twelve errors in a region is 1.43×10^{-10} , which indicates that the SEC cache configuration is adequate.

With 256-set/4-way set-associative SEC cache and the target retention error rate set to 10^{-6} , the total SEC cache size is 36KB. According to CACTI 5.3 [136], assuming 45nm technology, the area of the SEC cache is $0.327mm^2$, the access latency is 0.683ns that is about 2 processor cycles when the clock rate is 2GHz, the leakage power is 25mW, and the energy consumption per access is 0.054nJ.

5.3.2.2 Energy Analysis

I first discuss the energy results of SECRET on DRAMs executing the workloads mentioned in Section 5.3.1. All the results reported here take all energy overheads dis-

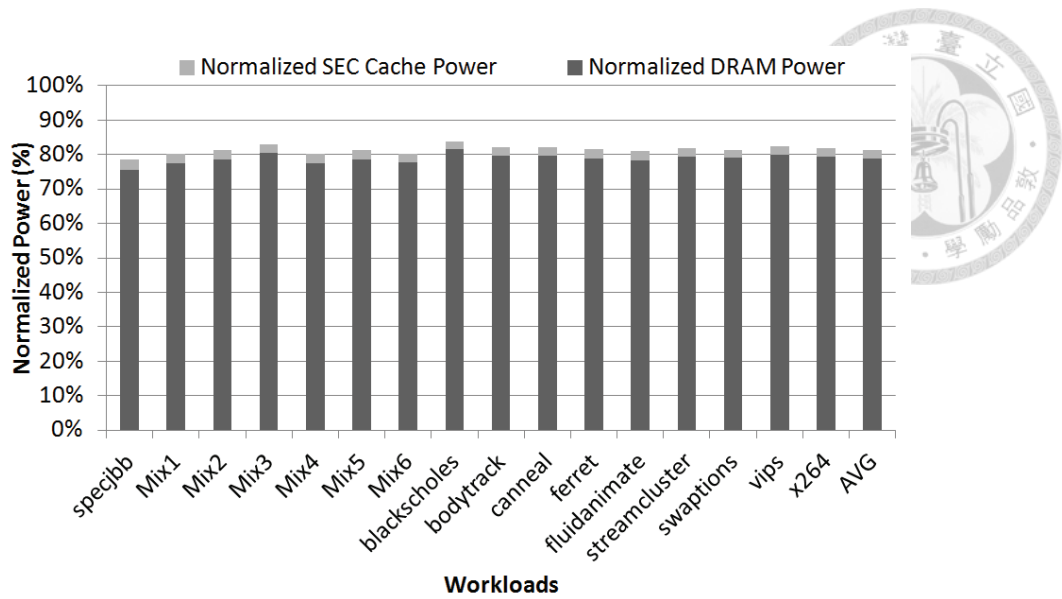


Figure 5.11: Power consumption of the DRAM system with SECRET normalized to the baseline.

cussed in Section 5.2.6 into consideration. Figure 5.11 shows the power consumption of the DRAM system with SECRET normalized to the baseline. The breakdown shows the power consumption of the SEC cache and the DRAM system, respectively. Even with an average of 2.6% more power overheads of the SEC cache, SECRET still achieves up to 18.57% DRAM power reduction on the average.

To see how SECRET affects the major parts of the DRAM power consumption, I breakdown the DRAM power consumption into DRAM peripheral leakage, dynamic and refresh power as shown in Figure 5.12. This set of results are normalized to the baseline. For each workload, the bar on the left and right are the results of baseline and SECRET, respectively. The DRAM refresh power is reduced by 87.2% for all workloads since the refresh interval is increased from 64ms to 500ms. In my experiments, when the power overheads are not considered, the refresh power reduction achieved by SECRET contributes 20.10% of total DRAM power reduction on the average. SECRET also achieves an average of 1.48% DRAM power reduction by reducing DRAM peripheral leakage power since infrequent refresh operations lead to long idle times. The dynamic power increases slightly, about 0.43% on the average, due to additional memory accesses for fetching the ECP directory and ECPs.

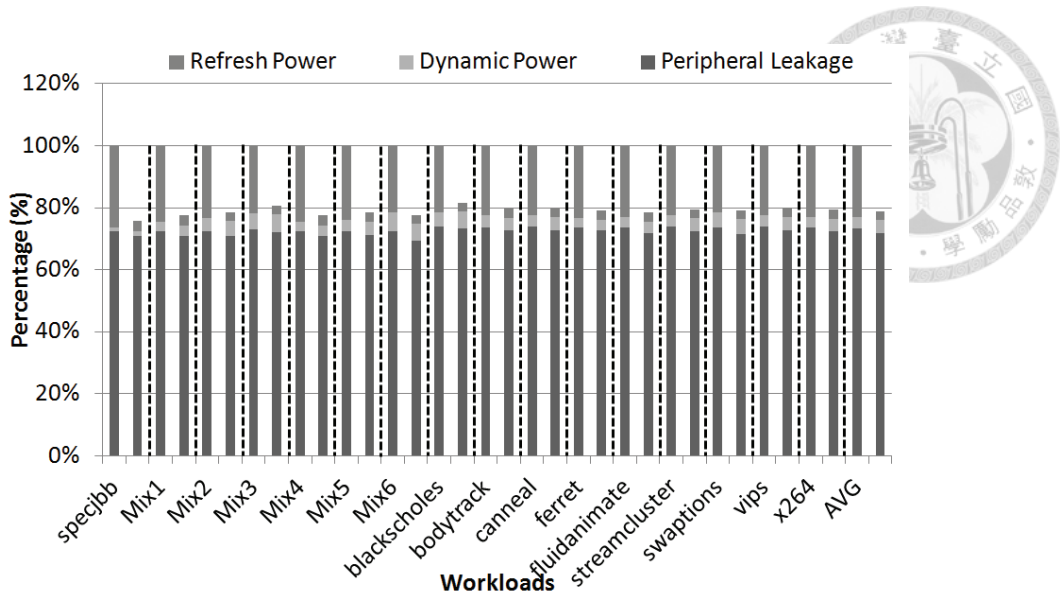


Figure 5.12: Power breakdown of DRAM peripheral leakage, dynamic and refresh power of the baseline (left) and SECRET (right).

5.3.2.3 Performance Analysis

As mentioned in Section 5.2.6, the performance overheads come from additional memory accesses caused by SEC cache misses. Figure 5.13 shows the breakdown of the three types of additional memory accesses, and the results are normalized to the number of data accesses issued by the workloads. We can see that SECRET introduces at most 2.87% and an average of 1.62% more data accesses in my tested cases. On the average, fetching the ECP directory introduces 0.84% more memory accesses, while fetching ECPs incurs 0.55% additional memory accesses. On an SEC cache miss, fetching the ECP directory is necessary, but fetching ECPs only happens when the requested region has retention errors. Therefore, the number of memory accesses for ECP directory is slightly more than that for ECPs. Writing ECPs back introduces only an average of 0.23% more memory accesses. The additional memory accesses do not incur noticeable penalty on overall performance. The average IPC values of the baseline and SECRET are 13.472 and 13.475, respectively. Among all test cases, x.264 has the most performance degradation when SECRET is applied. However, even in this case, x.264 only has 1.3% performance degradation compared to the baseline. For the workload Mix6, SECRET even achieves 1.4% performance improvement since DRAM modules are less likely to be occupied by refresh operations that are executed infrequently.

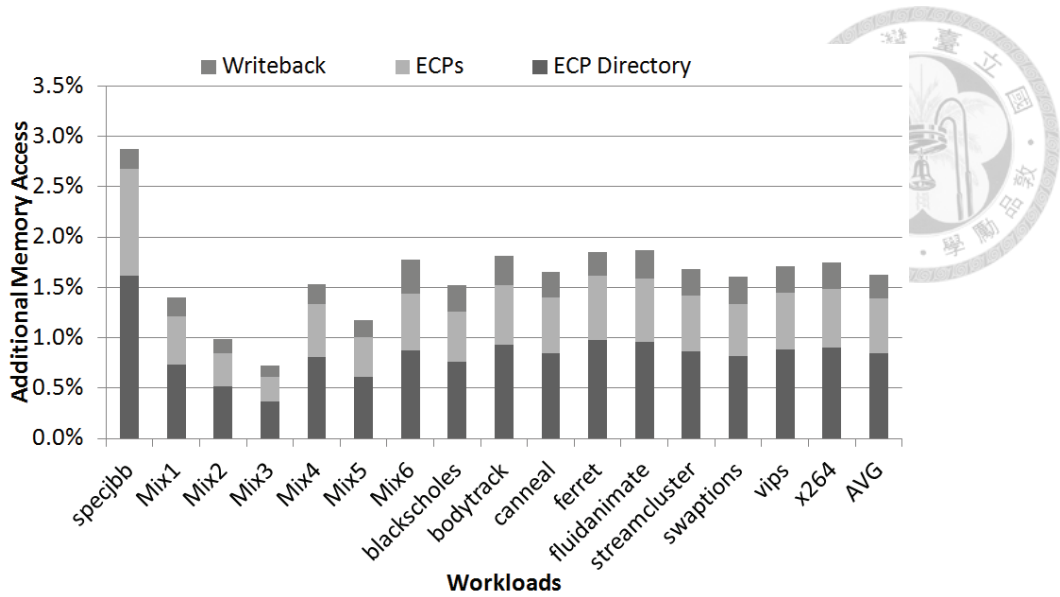


Figure 5.13: Additional memory accesses of the SECRET framework normalized to the number of data accesses issued by the workloads.

5.3.2.4 Evaluating SECRET with Reduced Last-level Cache Size

To see the performance of SECRET with DRAMs that have high access density and low percentage of refresh power consumption, I perform a set of experiments that have L2 cache sizes reduced from 8MB to 4MB and 2MB to generate more DRAM accesses. My experimental results show that, compared to the 8MB L2 cache, the 4MB and 2MB L2 caches respectively have 56.5% and 192.6% more DRAM accesses on the average. As mentioned in Section 5.2.3, the design of the SEC cache is decided during system design time along with the configuration of the micro-architecture. So, for the 4MB and 2MB L2 cache configurations, I respectively perform the process of design space exploration for the SEC cache as described in Section 5.3.2.1. The results show that the settings of 256-set/4-way SEC cache and 10^{-6} retention error rate are also adequate for systems with 4MB L2 cache. For the 2MB L2 cache configuration, since more DRAM accesses are introduced and the spatial locality of these accesses increases, the settings of 128-set/4-way SEC cache and 10^{-6} retention error rate are adequate.

Figure 5.14 shows the power breakdown of the DRAM systems that utilizing 8MB, 4MB and 2MB L2 caches. I show the average DRAM system power consumption of the 16 workloads both with and without SECRET. The results are normalized to the power consumption of the DRAM system with 8MB L2 cache without SECRET. Since the re-

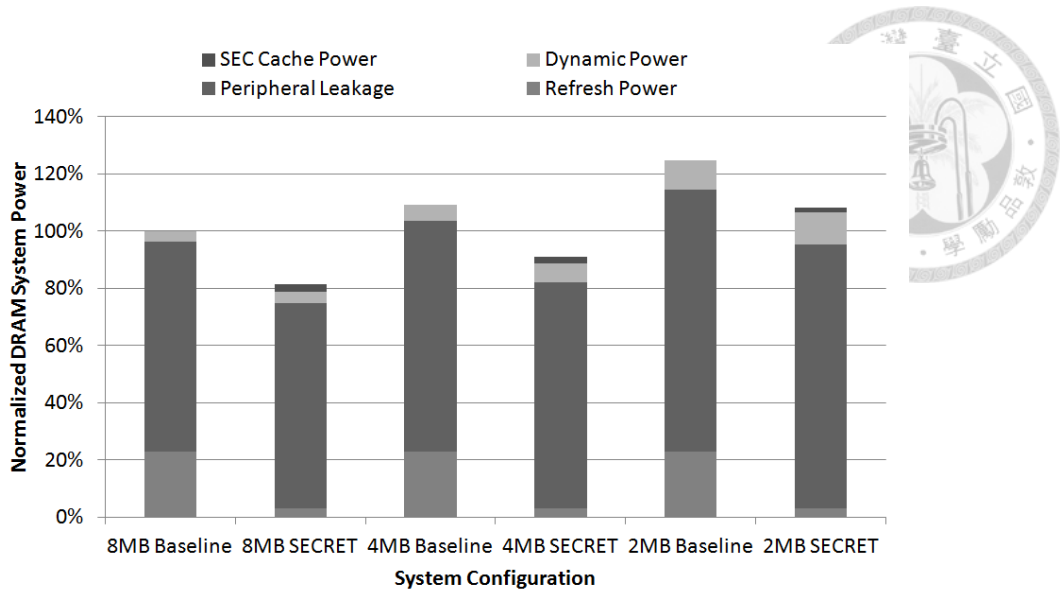


Figure 5.14: DRAM system power breakdown normalized to the DRAM power consumption with 8MB L2 cache.

fresh interval is increased from 64ms to 500ms, the DRAM refresh power is reduced by 87.2% for all the cases. However, with the decreasing L2 cache size and the increasing number of DRAM accesses, the percentage of dynamic power and peripheral leakage power also increase. Therefore, the percentage of DRAM power reduction by SECRET also diminishes with the L2 cache size. For the 8MB L2 cache configuration, SECRET achieves 18.57% DRAM power reduction. With the 4MB L2 cache configuration, the DRAM power reduction achieved by SECRET is reduced to 16.71%. However, even with the 2MB L2 cache that has 0.6% more percentage of peripheral leakage power and 5.1% more percentage of dynamic power than the 8MB L2 cache size, the SECRET can still achieve 13.5% DRAM power reduction on the average with 1.4% power overheads of the SEC cache.

Figure 5.15 shows the additional memory accesses introduced by SECRET when utilizing the three L2 cache sizes. As mentioned earlier, 256-set/4-way SEC cache size is utilized for both the 8MB and 4MB L2 caches. 128-set/4-way SEC cache size is utilized by the 2MB L2 cache. Although smaller L2 caches introduce more accesses to DRAMs and the SEC cache, the spatial locality of the accesses actually increases. Therefore, for the 4MB L2 cache configuration, it has lower SEC cache miss rate and less additional DRAM accesses than the 8MB L2 cache configuration even if systems with 4MB and

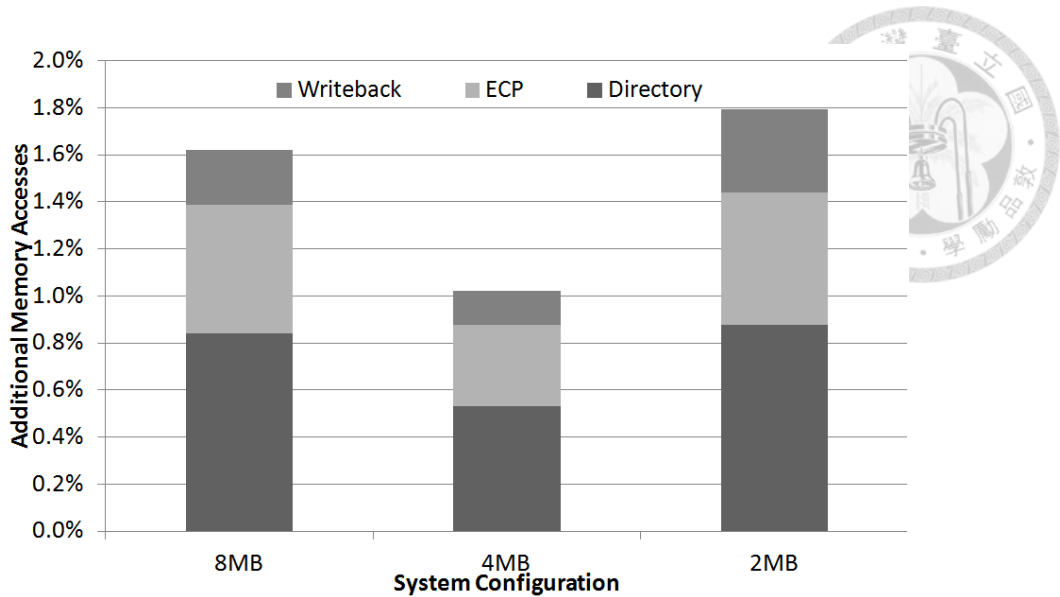


Figure 5.15: Average additional memory accesses of the SECRET framework normalized to the number of data accesses issued by the workloads with 8MB, 4MB and 2MB L2 cache.

8MB L2 caches both utilize the same SEC cache configuration. My experimental results show that, SECRET introduces very negligible performance degradation (less than 1%) for all L2 cache sizes.

5.3.2.5 Evaluation of Distributing Leaky Cells with Spatial Locality

As mentioned in Section 5.2.3, I assume the retention times of DRAM cells are uniformly and randomly distributed. Based on this assumption, the SEC cache line size is decided. However, when the clustering of leaky cells happens, the SECRET framework would choose a shorter refresh interval so that the maximum number of retention errors in a region can be fit in the selected SEC cache line size. In the end, the amount of refresh power reduction may be reduced. So, in this set of experiments, I create a set of retention time distributions with various degrees of spatial locality to evaluate SECRET. Distributions with high spatial locality indicate that leaky cells are more likely to be clustered in the neighboring area.

According to [137], the retention time of a DRAM cell, denoted by $T_{retention}$, can be decided by two kinds of effects, random effects and the systematic effects, where the systematic effects refer to the layout-dependent variation through which nearby devices

share similar parameters. Therefore, $T_{retention}$ is modeled by

$$T_{retention} = (1 - W) \cdot T_{rand} + W \cdot T_{sys}, \quad (5.6)$$

where T_{rand} denotes the retention time decided by random effects, and T_{sys} denotes the retention time decided by systematic effects. W is the weight value to adjust the proportion between random and systematic effects. When W is set to zero, this indicates I consider the random effect only and no spatial locality among leaky cells when generating the retention time distribution. For this set of experiments, I randomly generate T_{rand} . For generating T_{sys} , I adopt the multiple-level quad tree approach [138] to model the correlated within-die variation effect among retention times of DRAM cells. The smallest quadrant in the multiple-level quad tree is set to 16K DRAM cells [137].

In this set of experiments, I set W to 0.0, 0.1, 0.2, 0.3, 0.4 and 0.5 to model the distributions with various degrees of spatial locality. For each W value, I utilize Equation 5.6 to generate ten sets of retention time distributions. Therefore, there are sixty different retention time distributions generated for this set of experiments. The L2 cache size is set to 8MB. The settings of SEC cache and the target retention error rate are the same as the ones obtained in Section 5.3.2.1. That is, the SEC cache line size is set to 36 bytes, which is able to store at most twelve ECPs of a region, and the target error rate is set to 10^{-6} .

For the retention time distributions with various W values, in Figure 5.16, I show the DRAM power reduction of various distributions. Moreover, it also shows the maximum number of retention errors in a region when the target error rate is set to 10^{-6} . We can observe that, with the value of W increases, the spatial locality among retention times slightly increases and the maximum number of errors in a region also increases. However, for the sixty retention time distributions, only four distributions have the maximum number of retention errors larger than twelve when the target error rate is set to 10^{-6} in my test cases. To accommodate the selected SEC cache design, instead of utilizing the 500ms refresh interval, the refresh intervals of the four cases need to be shortened so that the maximum number of retention errors in a region is no more than twelve. The refresh interval that meets the requirement of each of the four cases is also marked in Figure 5.16, and the shortest refresh interval is 450ms. Although shorter refresh interval indicates less refresh power reduction, the 450ms refresh interval achieves only 0.8% less power reduction than the best case. This shows the proposed SECRET framework can still achieve

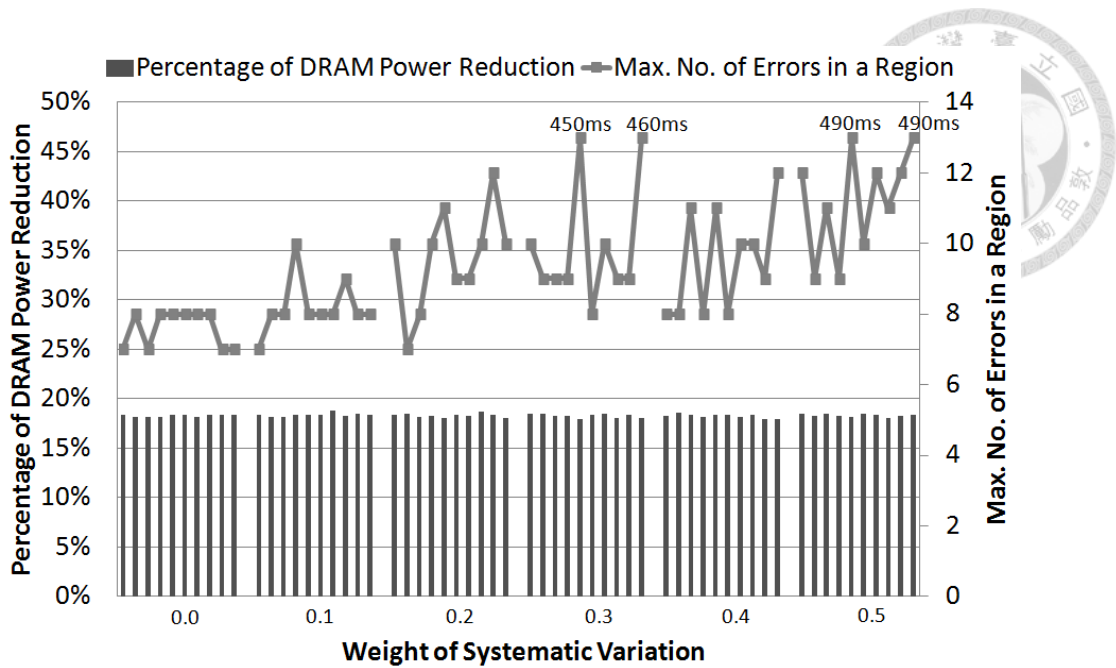


Figure 5.16: DRAM power reduction of various distributions, and the maximum number of retention errors in a region with the target error rate set to 10^{-6} .

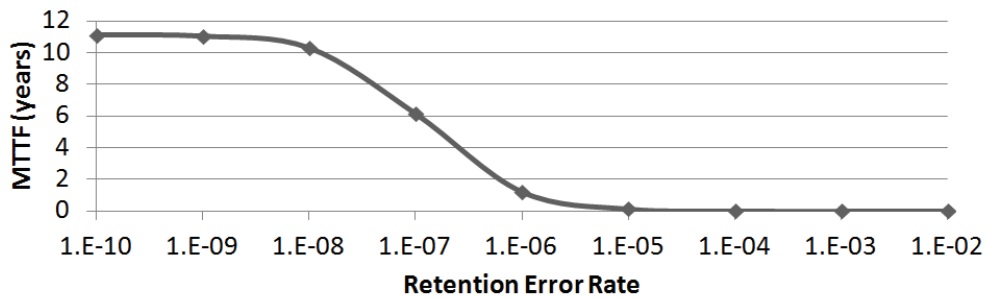


Figure 5.17: Mean time to failure vs. retention error rate in the DRAM system with Hamming Code ECC.

significant DRAM power reduction when extreme cases of the retention time distribution happens.

5.3.2.6 Comparison with Traditional ECC Approaches

For systems with a reliability requirement, DRAMs are commonly protected by the Hamming Code ECC to meet the target MTTF. Here, I evaluate the effect of SECRET in such a DRAM system. With the same ECC capability, to tolerate retention errors caused by prolonging the refresh interval, the MTTF of the protected memory system is shortened.

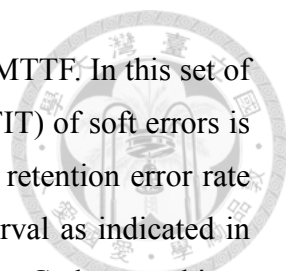


Figure 5.17 shows the relation between retention error rates and the MTTF. In this set of experiments, I assume a 64GB DRAM, where the Failure in Time (FIT) of soft errors is 2000/Mb [139]. To meet the 10-year MTTF requirement [140], the retention error rate should not exceed 10^{-8} , which corresponds to a 128ms refresh interval as indicated in Figure 5.7. That is, under the MTTF constraint, DRAMs with Hamming Code can achieve 50% refresh power reduction compared to the baseline with a 64ms refresh interval. As shown in Section 5.3.2.2, SECRET can increase the refresh interval to 500ms and achieve up to 87.2% refresh power reduction. Therefore, the benefit of SECRET is still quite substantial in a DRAM system protected with Hamming Code.

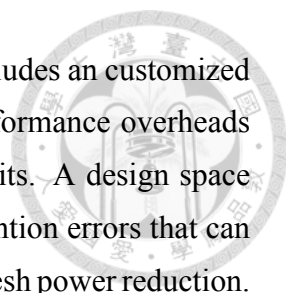


Chapter 6

Conclusion

In this dissertation, I develop the PPT framework and the SECRET framework to reduce DRAM background power. The PPT framework is the first joint performance, power and thermal (PPT) management framework to orchestrate task execution and page allocation among tasks to achieve desirable tradeoff between performance, power and temperature. Previous power-aware DRAM system designs cluster memory accesses to prolong the idle periods, so they can utilize low-power modes to reduce DRAM peripheral leakage efficiently. However, these mechanisms may increase the power density of the active DRAM modules and cause thermal emergency. The PPT framework also reduces the DRAM peripheral leakage by clustering memory accesses, but controls the operating temperature by alternating active DRAM modules periodically. With accurate bandwidth demand estimation, the PPT framework can allocate sufficient memory resources for the running threads to avoid performance degradation due to clustering memory accesses into a subset of DRAM modules. The experimental results show that PPT with the dynamic BW estimation method achieves comparable performance to the performance-driven policy at all time, but much lower power consumption, while the power-driven policy could only deliver half of the peak throughput of PPT. Furthermore, PPT has the lowest temperature compared to the performance-driven and power-driven policies.

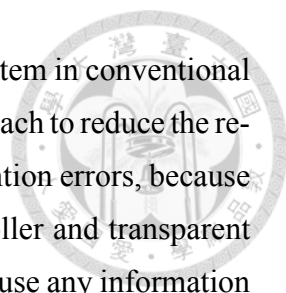
The SECRET framework is a novel error correction framework for retention errors in DRAMs to prolong the refresh interval and achieve refresh power reduction based on the new observations that the retention errors are hard errors rather than soft errors and only few DRAM cells have large leakage. Therefore, SECRET only allocates error-correcting bits to leaky cells that occur retention errors under a refresh interval. So, the SECRET framework is much more area and energy efficient than conventional ECC schemes for



refresh power reduction. Moreover, the SECRET framework also includes an customized cache for the selective error correction scheme to minimize the performance overheads from the additional memory accesses for fetching error-correcting bits. A design space exploration process is also proposed to select a target number of retention errors that can achieve the best balance between the correction overheads and the refresh power reduction. SECRET does not require any modification in the interface between the memory controller and the DRAM, and all additional hardware components are in the memory controller. The experimental results show that SECRET can reduce refresh power by 87.2%, and an average of 18.57% of DRAM power consumption with negligible area and performance overheads. The results also show that, for DRAM systems that suffer spatial locality of retention time distribution or for DRAM systems protected by Hamming Code, the benefit of SECRET is still quite substantial.

The PPT framework reduces the peripheral leakage power consumption of DRAM systems by utilizing the low-power modes with power-aware page allocation and thread scheduling, and the mechanism is mainly implemented in the system software such as the operating system. So, the PPT framework is a software approach with hardware supports to achieve power reduction. On the other hand, the SECRET framework reduces the refresh power consumption of DRAM systems by prolonging the refresh interval and correcting retention errors, and the SEC mechanism is a hardware approach that is implemented in the memory controller and transparent to the system software and applications. So, the SECRET framework is a hardware approach with few software supports to achieve power savings. Since they both incur negligible performance impact for the DRAM systems and reduce different parts of the DRAM background power consumption with different approaches, they can be used on a DRAM system at the same time to achieve power reduction on both peripheral leakage and refresh power of the DRAM system.

I choose to use a software approach to reduce peripheral leakage power, because the PPT framework needs to manipulate the memory access behaviors, and the hardware approaches are usually not able to collect enough information to perform an efficient coarse-grained reshaping of memory accesses. For example, the hardware approaches may not be able to perform power-aware data allocation or power-aware memory access scheduling as efficiently as the mechanisms implemented in the operating system by utilizing power-aware page allocation and power-aware thread scheduling, because the page



allocation and thread scheduling are already done by the operating system in conventional computer systems. On the other hand, I choose to use a hardware approach to reduce the refresh power by prolonging the refresh interval and correcting the retention errors, because the refresh operations are basically controlled by the memory controller and transparent to the system software and applications. Since this approach does not use any information from the operating system or the executed applications, there is no need to implement this mechanism in software by increasing the design complexity of the operating system or changing the applications.

However, to reduce the peripheral leakage power, the power-aware page allocation or thread scheduling policies may be able to get some advantages from utilizing hardware supports. For example, the dynamic bandwidth estimation method of the *Adaptive Grouping* mechanism in the PPT framework uses the usage of the memory request buffer to direct the *Adaptive Grouping* mechanism and shows more power savings than the static bandwidth estimation method that is a software approach. On the other hand, to reduce the refresh power, the hardware approaches such as the SECRET framework can cooperate with the software approaches such as previous works aforementioned in Section 2.2.3 to further reduce refresh operations. Sometimes, the software approaches are better, because the operating system and applications can provide a high-level overall picture about the system behaviors and pure software approaches can avoid the overheads of modifications in hardware. But, the hardware approaches are sometimes better, because hardware components can perform fine-grained and instant response for specific system behaviors and pure hardware approaches can avoid the overheads of re-design or re-compilation of the operating system and applications. However, in general, the software-hardware cooperative approaches can get more information than the software-only or hardware-only approaches and provide more opportunities to save energy at the expense of more design complexity. When we design the low-power policies, we need to know the complexity of retrieving the required information and the overheads of implementing the low-power mechanism, and try to figure out how to change the system can minimize the design complexity and overheads by utilizing software approaches, hardware approaches or software-hardware cooperative approaches. To minimize the DRAM background power composed of the peripheral leakage power and the refresh power, I believe that putting the PPT and SECRET frameworks together is a good approach to build a software-hardware cooper-

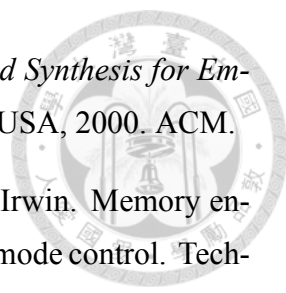
ative mechanism that efficiently reduces the DRAM background power with acceptable implementation overheads.

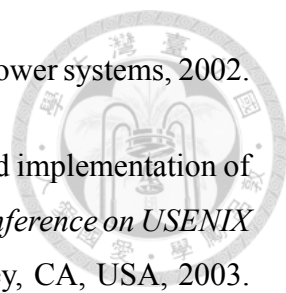


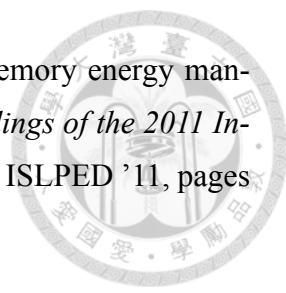
Bibliography

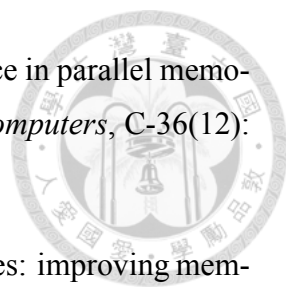



- [1] Stuart Schechter, Gabriel H. Loh, Karin Straus, and Doug Burger. Use ecp, not ecc, for hard failures in resistive memories. In *Proceedings of the 37th Annual International Symposium on Computer Architecture, ISCA '10*, pages 141–152, New York, NY, USA, 2010. ACM.
- [2] Marc A. Viredaz and Deborah A. Wallach. Power evaluation of a handheld computer: A case study. Technical report, Compaq Western Research Laboratory, 2001.
- [3] Charles Lefurgy, Karthick Rajamani, Freeman Rawson, Wes Felter, Michael Kistler, and Tom W. Keller. Energy management for commercial servers. *Computer*, 36(12):39–48, December 2003.
- [4] Odilio Vargas. Minimum power consumption in mobile-phone memory subsystems. *Portable Design Magazine*, 26:31–38, 2005.
- [5] Odilio Vargas. Achieve minimum power consumption in mobile memory subsystems. *EE Times Asia*, 2006.
- [6] Elliott Cooper-Balis and Bruce Jacob. Fine-grained activation for power reduction in dram. *IEEE Micro*, 30(3):34–47, May 2010.
- [7] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. Raidr: Retention-aware intelligent dram refresh. In *Proceedings of the 39th Annual International Symposium on Computer Architecture, ISCA '12*, pages 1–12, Washington, DC, USA, 2012. IEEE Computer Society.
- [8] V. Delaluz, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Energy-oriented compiler optimizations for partitioned memory architectures. In *Proceedings of the*

- 
- 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '00, pages 138–147, New York, NY, USA, 2000. ACM.
- [9] V. Delaluz, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Memory energy management using software and hardware directed power mode control. Technical report, 2000.
- [10] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '00, pages 105–116, New York, NY, USA, 2000. ACM.
- [11] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M.J. Irwin. Dram energy management using software and hardware directed power mode control. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 159–169, 2001.
- [12] V. Delaluz, M. Kandemir, N. Vijaykrishnan, Anand Sivasubramaniam, and M.J. Irwin. Hardware and software techniques for controlling dram power modes. *IEEE Transactions on Computers*, 50(11):1154–1173, 2001.
- [13] R. Athavale, Narayanan Vijaykrishnan, Mahmut T. Kandemir, and Mary Jane Irwin. Influence of array allocation mechanisms on memory system energy. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, IPDPS '01, pages 3–, Washington, DC, USA, 2001. IEEE Computer Society.
- [14] V. De La Luz, M. Kandemir, and I. Kolcu. Automatic data migration for reducing energy consumption in multi-bank memory systems. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 213–218, New York, NY, USA, 2002. ACM.
- [15] V. Delaluz, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, A. Sivasubramaniam, and I. Kolcu. Compiler-directed array interleaving for reducing energy in multi-bank memories. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, ASP-DAC '02, pages 288–, Washington, DC, USA, 2002. IEEE Computer Society.

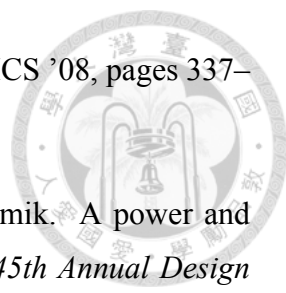
- 
- [16] N. Tuck and S. Reda. Os energy conservation policies for low power systems, 2002.
- [17] Hai Huang, Padmanabhan Pillai, and Kang G. Shin. Design and implementation of power-aware virtual memory. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference, ATEC '03*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.
- [18] Zhong Wang and Xiaobo Sharon Hu. Power aware variable partitioning and instruction scheduling for multiple memory banks. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '04*, pages 10312–, Washington, DC, USA, 2004. IEEE Computer Society.
- [19] Zhong Wang and Xiaobo Sharon Hu. Energy-aware variable partitioning and instruction scheduling for multibank memory architectures. *ACM Transactions on Design Automation of Electronic Systems*, 10(2):369–388, April 2005.
- [20] Hai Huang, Kang G. Shin, Charles Lefurgy, and Tom Keller. Improving energy efficiency by making dram less randomly accessed. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design, ISLPED '05*, pages 393–398, New York, NY, USA, 2005. ACM.
- [21] Ozcan Ozturk and Mahmut Kandemir. Integer linear programming based energy optimization for banked drams. In *Proceedings of the 15th ACM Great Lakes Symposium on VLSI, GLSVLSI '05*, pages 92–95, New York, NY, USA, 2005. ACM.
- [22] Ozcan Ozturk and Mahmut Kandemir. Nonuniform banking for reducing memory energy consumption. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE '05*, pages 814–819, Washington, DC, USA, 2005. IEEE Computer Society.
- [23] Vivek Pandey, W. Jiang, Yuanyuan Zhou, and R. Bianchini. Dma-aware memory energy management. In *Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture, HPCA '06*, pages 133–144, 2006.

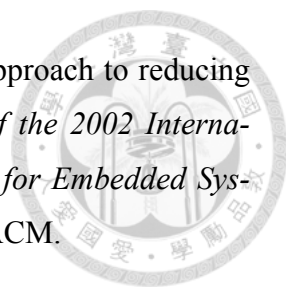
- 
- [24] K. Kumar, K. Doshi, M. Dimitrov, and Yung-Hsiang Lu. Memory energy management for an enterprise decision support system. In *Proceedings of the 2011 International Symposium on Low Power Electronics and Design, ISLPED '11*, pages 277–282, 2011.
- [25] R. Ayoub, R. Nath, and T. Rosing. Jetc: Joint energy thermal and cooling management for memory and cpu subsystems in servers. In *Proceedings of the 2012 18th IEEE International Symposium on High Performance Computer Architecture, HPCA '12*, pages 1–12, 2012.
- [26] Ahmed M. Amin and Zeshan A. Chishti. Rank-aware cache replacement and write buffering to improve dram energy efficiency. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '10*, pages 383–388, New York, NY, USA, 2010. ACM.
- [27] Victor De La Luz, Ismail Kadayif, Mahmut Kandemir, and Uger Sezer. Access pattern restructuring for memory energy. *IEEE Transactions on Parallel and Distributed Systems*, 15(4):289–303, April 2004.
- [28] I. Hur and C. Lin. A comprehensive approach to dram power management. In *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture, HPCA '08*, pages 305–316, 2008.
- [29] Krishna Kant. A control scheme for batching dram requests to improve power efficiency. *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, 39(1):331–332, June 2011.
- [30] Sukki Kim, Soontae Kim, and Yebin Lee. Dram power-aware rank scheduling. In *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '12*, pages 397–402, New York, NY, USA, 2012. ACM.
- [31] O. Ozturk, G. Chen, M. Kandemir, and M. Karakoy. Cache miss clustering for banked memory systems. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '06*, pages 244–250, New York, NY, USA, 2006. ACM.

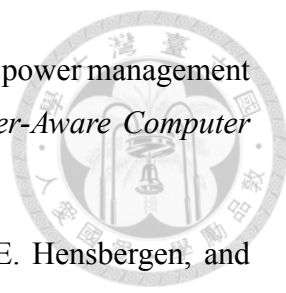
- 
- [32] David T. Harper and J. Robert Jump. Vector access performance in parallel memories using a skewed storage scheme. *IEEE Transactions on Computers*, C-36(12): 1440–1449, 1987.
- [33] André Seznec and Jacques Lenfant. Interleaved parallel schemes: improving memory throughput on supercomputers. In *Proceedings of the 19th Annual International Symposium on Computer Architecture, ISCA '92*, pages 246–255, New York, NY, USA, 1992. ACM.
- [34] Scott Rixner. Memory controller optimizations for web servers. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '04*, pages 355–366, Washington, DC, USA, 2004. IEEE Computer Society.
- [35] Jungeun Kim and Taewhan Kim. Memory access optimization through combined code scheduling, memory allocation, and array binding in embedded system design. In *Proceedings of the 42nd Annual Design Automation Conference, DAC '05*, pages 105–110, New York, NY, USA, 2005. ACM.
- [36] I-Jui Sung, John A. Stratton, and Wen-Mei W. Hwu. Data layout transformation exploiting memory-level parallelism in structured grid many-core applications. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT '10*, pages 513–522, New York, NY, USA, 2010. ACM.
- [37] Y. Ben Asher and N. Rotem. Automatic memory partitioning: Increasing memory parallelism via data structure partitioning. In *Proceedings of the 2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '10*, pages 155–161, 2010.
- [38] Ciprian Seiculescu, Luca Benini, and Giovanni De Micheli. A distributed interleaving scheme for efficient access to wideio dram memory. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS '12*, pages 103–112, New York, NY, USA, 2012. ACM.


- 
- [39] Lei Liu, Zehan Cui, Mingjie Xing, Yungang Bao, Mingyu Chen, and Chengyong Wu. A software memory partition approach for eliminating bank-level interference in multicore systems. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 367–376, New York, NY, USA, 2012. ACM.
- [40] Heekwon Park, Seungjae Baek, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Regularities considered harmful: forcing randomness to memory accesses to reduce row buffer conflicts for multi-core, multi-bank systems. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, pages 181–192, New York, NY, USA, 2013. ACM.
- [41] Joonghyun Baek, Byungse So, Taekoo Lee, Yunhyeok Im, and Seyong Oh. Thermal characterization of high speed ddr devices in system environments. In *Proceedings of the Nineteenth Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, SEMI-THERM '03, pages 138–143, 2003.
- [42] J Iyer, CL Hall, J Shi, and Y Huang. System memory power and thermal management in platforms built on intel centrino duo mobile technology. *Intel Technology Journal*, 10(2):123–132, 2006.
- [43] H. Ran and I. Mohammed. Thermal management of high density very low profile memory module. In *Proceedings of the Twenty Third Annual IEEE Semiconductor Thermal Measurement and Management Symposium*, SEMI-THERM '07, pages 118–124, 2007.
- [44] Jiang Lin, Hongzhong Zheng, Zhichun Zhu, Howard David, and Zhao Zhang. Thermal modeling and management of dram memory systems. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 312–322, New York, NY, USA, 2007. ACM.
- [45] Jiang Lin, Hongzhong Zheng, Zhichun Zhu, Eugene Gorbатов, Howard David, and Zhao Zhang. Software thermal management of dram memory for multicore systems. In *Proceedings of the 2008 ACM SIGMETRICS International Conference on*

Measurement and Modeling of Computer Systems, SIGMETRICS '08, pages 337–348, New York, NY, USA, 2008. ACM.

- 
- [46] Song Liu, Seda Ogrenci Memik, Yu Zhang, and Gokhan Memik. A power and temperature aware dram architecture. In *Proceedings of the 45th Annual Design Automation Conference*, DAC '08, pages 878–883, New York, NY, USA, 2008. ACM.
- [47] Song Liu, Seda Ogrenci Memik, Yu Zhang, and Gokhan Memik. An approach for adaptive dram temperature and power management. In *Proceedings of the 22nd Annual International Conference on Supercomputing*, ICS '08, pages 63–72, New York, NY, USA, 2008. ACM.
- [48] Raid Zuhair Ayoub, Krishnam Raju Indukuri, and Tajana Simunic Rosing. Energy efficient proactive thermal management in memory subsystem. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, pages 195–200, New York, NY, USA, 2010. ACM.
- [49] Song Liu, B. Leung, A. Neekar, S.O. Memik, G. Memik, and N. Hardavellas. Hardware/software techniques for dram thermal management. In *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture*, HPCA '11, pages 515–525, 2011.
- [50] Ozcan Ozturk and Mahmut Kandemir. Data replication in banked drams for reducing energy consumption. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, ISQED '06, pages 551–556, Washington, DC, USA, 2006. IEEE Computer Society.
- [51] Renato Levy, Bhagirath Narahari, and Rahul Simha. Energy-aware allocation of dynamic variables in partitioned memory architectures.
- [52] A.E. Papathanasiou and M.L. Scott. Energy efficiency through burstiness. In *Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems and Applications*, HotMobile '03, pages 44–53, 2003.

- 
- [53] Jayaprakash Pisharath and Alok Choudhary. An integrated approach to reducing power dissipation in memory hierarchies. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '02, pages 88–97, New York, NY, USA, 2002. ACM.
- [54] H. Koc, O. Ozturk, M. Kandemir, S. H. K. Narayanan, and E. Ercanli. Minimizing energy consumption of banked memories using data recomputation. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design*, ISLPED '06, pages 358–362, New York, NY, USA, 2006. ACM.
- [55] Xiaobo Fan, Carla Ellis, and Alvin Lebeck. Memory controller policies for dram power management. In *Proceedings of the 2001 International Symposium on Low Power Electronics and Design*, ISLPED '01, pages 129–134, New York, NY, USA, 2001. ACM.
- [56] V. Delaluz, A. Sivasubramaniam, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based dram energy management. In *Proceedings of the 39th Annual Design Automation Conference*, DAC '02, pages 697–702, New York, NY, USA, 2002. ACM.
- [57] Jayaprakash Pisharath, Alok Choudhary, and Mahmut Kandemir. Reducing energy consumption of queries in memory-resident database systems. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '04, pages 35–45, New York, NY, USA, 2004. ACM.
- [58] Chun-Gi Lyuh and Taewhan Kim. Memory access scheduling and binding considering energy minimization in multi-bank memory systems. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 81–86, New York, NY, USA, 2004. ACM.
- [59] C. G Lyuh and T. Kim. Memory access scheduling and binding considering energy minimisation in multi-bank memory systems: integrated approach. *IEE Proceedings - Computers and Digital Techniques*, 153(1):59–68, 2006.

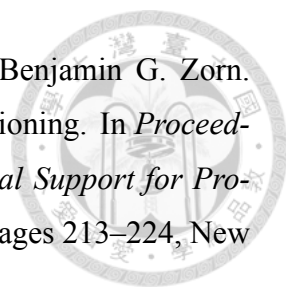
- 
- [60] Hai Huang and Kang G. Shin. Co-operative software-hardware power management for main memory. In *Proceedings of the Workshop on Power-Aware Computer Systems*, PACS '04, 2004.
- [61] H. Huang, K. G. Shin, C. Lefurgy, K. Rajamani, T. Keller, E. Hensbergen, and F. Rawson. Software-hardware cooperative power management for main memory. In *Proceedings of the 4th International Conference on Power-Aware Computer Systems*, PACS '04, pages 61–77, Berlin, Heidelberg, 2005. Springer-Verlag.
- [62] Xiaodong Li, Zhenmin Li, Yuanyuan Zhou, and Sarita Adve. Performance directed energy management for main memory and disks. *ACM Transactions on Storage*, 1(3):346–380, August 2005.
- [63] Hongzhong Zheng and Zhichun Zhu. Power and performance trade-offs in contemporary dram system designs for multicore processors. *IEEE Transactions on Computers*, 59(8):1033–1046, August 2010.
- [64] Mingsong Bi, R. Duan, and C. Gniady. Delay-hiding energy management mechanisms for dram. In *Proceedings of the 16th IEEE International Symposium on High Performance Computer Architecture*, HPCA '10, pages 1–10, 2010.
- [65] Bruno Zatt, Muhammad Shafique, Sergio Bampi, and Jörg Henkel. A low-power memory architecture with application-aware power management for motion & disparity estimation in multiview video coding. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, pages 40–47, Piscataway, NJ, USA, 2011. IEEE Press.
- [66] Muhammad Shafique, Bruno Zatt, Fabio Leandro Walter, Sergio Bampi, and Jörg Henkel. Adaptive power management of on-chip video memory for multiview video coding. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 866–875, New York, NY, USA, 2012. ACM.
- [67] J. Mukundan and J.F. Martinez. Morse: Multi-objective reconfigurable self-optimizing memory scheduler. In *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture*, HPCA '12, pages 1–12, 2012.

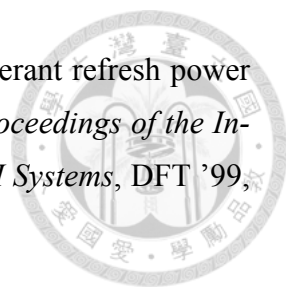
- 
- [68] Karthik Chandrasekar, Benny Akesson, and Kees Goossens. Run-time power-down strategies for real-time sdr memory controllers. In *Proceedings of the 49th Annual Design Automation Conference, DAC '12*, pages 988–993, New York, NY, USA, 2012. ACM.
- [69] Jabulani Nyathi and J.G. Delgado-Frias. Self-timed refreshing approach for dynamic memories. In *Proceedings of the Eleventh Annual IEEE International ASIC Conference*, pages 169–173, 1998.
- [70] H.Y. Cho and J.K. Oh. Us patent 6229747 b1: Self-refresh apparatus for a semiconductor memory device, May 2001.
- [71] L.L.C. Hsu, G. Frankowsky, and O. Weinfurter. Us patent 6483764 b2: Dynamic dram refresh rate adjustment based on cell leakage monitoring, November 2002.
- [72] M.J. Burgan. Wo patent 2004075256 a2: Variable refresh control for a memory, September 2004.
- [73] Tung-Han Tsai, Chin-Lin Chen, Ching-Li Lee, and Chua-Chin Wang. Power-saving nano-scale drams with an adaptive refreshing clock generator. In *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS '08*, pages 612–615, 2008.
- [74] Le-Nguyen Tran, F.J. Kurdahi, A.M. Eltawil, and A. Aljumah. Adjustable supply voltages and refresh cycle for process variations, temperature changes, and device degradation adaptation in 1t1c embedded dram. In *Proceedings of the 6th IEEE International Design and Test Workshop, IDT '11*, pages 124–129, 2011.
- [75] P.G. Emma, W.R. Reohr, and L.-K. Wang. Us patent 6389505 b1: Restore tracking system for dram, May 2002.
- [76] Mrinmoy Ghosh and Hsien Hsin S. Lee. Dram decay: Using decay counters to reduce energy consumption in drams.
- [77] Mrinmoy Ghosh and Hsien-Hsin S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams.

In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '07, pages 134–145, Washington, DC, USA, 2007. IEEE Computer Society.

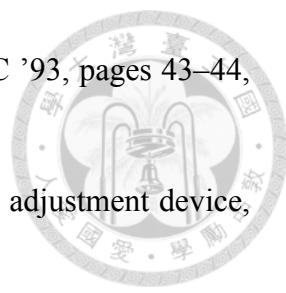


- [78] P.G. Emma, W.R. Reohr, and M. Meterelliyoz. Rethinking refresh: Increasing availability and reducing power in dram for cache applications. *IEEE Micro*, 28(6): 47–56, 2008.
- [79] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas. Refrint: Intelligent refresh to minimize power in on-chip multiprocessor cache hierarchies. In *Proceedings of the 19th International Symposium on High-Performance Computer Architecture*, HPCA '13, 2013.
- [80] T. Ohsawa, K. Kai, and K. Murakami. Optimizing the dram refresh count for merged dram/logic lsis. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design*, ISLPED '98, pages 82–87, 1998.
- [81] R.K. Venkatesan, S. Herr, and E. Rotenberg. Retention-aware placement in dram (rapid): software methods for quasi-non-volatile dram. In *Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture*, HPCA '06, pages 155–165, 2006.
- [82] V.G. Moshnyaga, Hua Vo, G. Reinman, and M. Potkonjak. Reducing energy of dram/flash memory system by os-controlled data refresh. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, ISCAS '07, pages 2108–2111, 2007.
- [83] Ciji Isen and Lizy John. Eskimo: Energy savings using semantic knowledge of inconsequential memory occupancy for dram subsystem. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '09, pages 337–346, New York, NY, USA, 2009. ACM.
- [84] S. Kim and S. Kim. Operating system-directed dram refresh energy reduction. In *Proceedings of the 6th Triangle Symposium on Advanced ICT*, TRISAI '11, 2011.

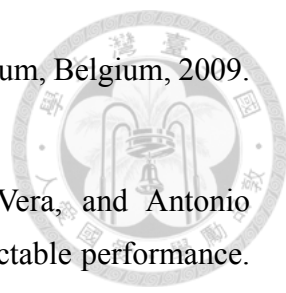
- 
- [85] Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flicker: saving dram refresh-power through critical data partitioning. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '11*, pages 213–224, New York, NY, USA, 2011. ACM.
- [86] K. Patel, L. Benini, Enrico Macii, and Massimo Poncino. Energy-efficient value-based selective refresh for embedded drams. In *Proceedings of the 15th International Conference on Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation, PATMOS '05*, pages 466–476, Berlin, Heidelberg, 2005. Springer-Verlag.
- [87] S. Takase and N. Kushiyama. A 1.6-gbyte/s dram with flexible mapping redundancy technique and additional refresh scheme. *IEEE Journal of Solid-State Circuits*, 34(11):1600–1606, 1999.
- [88] Joohee Kim and Marios C. Papaefthymiou. Dynamic memory design for low data-retention power. In *Proceedings of the 10th International Workshop on Integrated Circuit Design, Power and Timing Modeling, Optimization and Simulation, PATMOS '00*, pages 207–216, London, UK, UK, 2000. Springer-Verlag.
- [89] Joohee Kim and M.C. Papaefthymiou. Block-based multi-period refresh for energy efficient dynamic memory. In *Proceedings of the 14th Annual IEEE International ASIC/SOC Conference*, pages 193–197, 2001.
- [90] Joohee Kim and M.C. Papaefthymiou. Block-based multiperiod dynamic memory design for low data-retention power. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(6):1006–1018, 2003.
- [91] Todd Austin, Valeria Bertacco, David Blaauw, and Trevor Mudge. Opportunities and challenges for better than worst-case design. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference, ASP-DAC '05*, pages 2–7, New York, NY, USA, 2005. ACM.

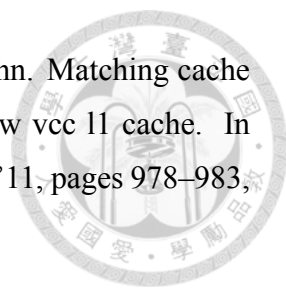
- 
- [92] Y. Katayama, E.J. Stuckey, S. Morioka, and Z. Wu. Fault-tolerant refresh power reduction of drams for quasi-nonvolatile data retention. In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems*, DFT '99, pages 311–318, 1999.
- [93] Yasunao Katayama, Yasushi Negishi, and Sumio Morioka. Efficient error correction code configurations for quasi-nonvolatile data retention by drams. In *Proceedings of the 15th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, DFT '00, pages 201–, Washington, DC, USA, 2000. IEEE Computer Society.
- [94] D.A. Klein and J. Schreck. Us patent 6965537 b1: Memory system and method using ecc to achieve low power refresh, November 2005.
- [95] Saeng-Hwan Kim, Won-Oh Lee, Jung-Ho Kim, Seong-Seop Lee, Sun-Young Hwang, Chang-Il Kim, Tae-Woo Kwon, Bong-Seok Han, Sung-Kwon Cho, Dae-Hui Kim, Jae-Keun Hong, Min-Yung Lee, Sung-Wook Yin, Hyeon-Gon Kim, Jin-Hong Ahn, Yong-Tark Kim, Yo-Hwan Koh, and Joong-Sik Kih. A low power and highly reliable 400mbps mobile ddr sdram with on-chip distributed ecc. In *Proceedings of the IEEE Asian Solid-State Circuits Conference*, ASSCC '07, pages 34–37, 2007.
- [96] S. Cha and H. Yoon. Check-bit-reduced codewords using non-2n data bits for ecc-based self-refresh enhancement techniques in drams. *Electronics Letters*, 46(22): 1488–1490, 2010.
- [97] Chris Wilkerson, Alaa R. Alameldeen, Zeshan Chishti, Wei Wu, Dinesh Somasekhar, and Shih-lien Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 83–93, New York, NY, USA, 2010. ACM.
- [98] T. Murotani. Us patent 4393477 a: Temperature responsive refresh control circuit, July 1983.
- [99] Y. Kagenishi, H. Hirano, A. Shibayama, H. Kotani, N. Moriwaki, M. Kojima, and T. Sumi. Low power self refresh mode dram with temperature detecting circuit. In

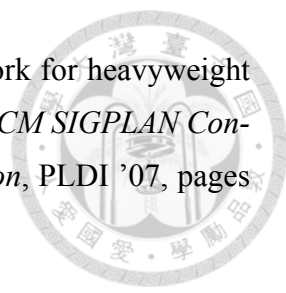
Proceedings of the IEEE Symposium on VLSI Circuits, VLSIC '93, pages 43–44, 1993.

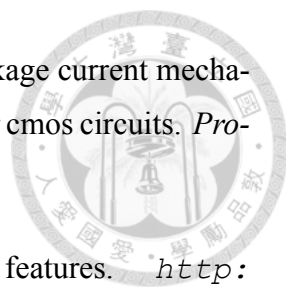
- 
- [100] H. Ruckerbauer. Us patent 6438057 b1: Dram refresh timing adjustment device, system and method, August 2002.
- [101] Elpida Memory, Inc. Auto temperature compensated self refresh (atcsr). 2005.
- [102] Chan-Kyung Kim, Jae-Goo Lee, Young-Hyun Jun, Chil-Gee Lee, and Bai-Sun Kong. Cmos temperature sensor with ring oscillator for mobile dram self-refresh control. *Microelectronics Journal*, 38(10-11):1042–1049, October 2007.
- [103] P.G. Emma and W. Roehr. Us patent 7483325 b2: Retention-time control and error management in a cache system comprising dynamic storage, January 2009.
- [104] Jie Meng, Katsutoshi Kawakami, and Ayse K. Coskun. Optimizing energy efficiency of 3-d multicore systems with stacked dram under power and thermal constraints. In *Proceedings of the 49th Annual Design Automation Conference, DAC '12*, pages 648–655, New York, NY, USA, 2012. ACM.
- [105] Jie Meng and A.K. Coskun. Analysis and runtime management of 3d systems with stacked dram for boosting energy efficiency. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition, DATE '12*, pages 611–616, 2012.
- [106] Chris Wilkerson, Hongliang Gao, Alaa R. Alameldeen, Zeshan Chishti, Muhammad Khellah, and Shih-Lien Lu. Trading off cache capacity for reliability to enable low voltage operation. In *Proceedings of the 35th Annual International Symposium on Computer Architecture, ISCA '08*, pages 203–214, Washington, DC, USA, 2008. IEEE Computer Society.
- [107] C. Wilkerson, Hongliang Gao, A.R. Alameldeen, Z. Chishti, M. Khellah, and Shih-Lien Lu. Trading off cache capacity for low-voltage operation. *IEEE Micro*, 29(1): 96–103, 2009.
- [108] Avesta Sasan (Mohammad A Makhzan), Houman Homayoun, Ahmed Eltawil, and Fadi Kurdahi. Process variation aware sram/cache for aggressive voltage-frequency scaling. In *Proceedings of the Conference on Design, Automation and*

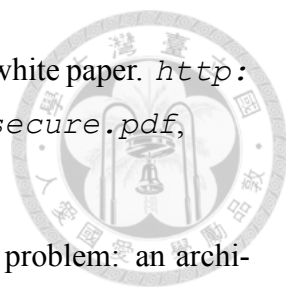
Test in Europe, DATE '09, pages 911–916, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

- 
- [109] Jaume Abella, Javier Carretero, Pedro Chaparro, Xavier Vera, and Antonio González. Low vccmin fault-tolerant cache with highly predictable performance. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '09, pages 111–121, New York, NY, USA, 2009. ACM.
- [110] Amin Ansari, Shuguang Feng, Shantanu Gupta, and Scott Mahlke. Enabling ultra low voltage system operation by tolerating on-chip cache failures. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, pages 307–310, New York, NY, USA, 2009. ACM.
- [111] Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Wei Wu, and Shih-Lien Lu. Improving cache lifetime reliability at ultra-low voltages. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '09, pages 89–99, New York, NY, USA, 2009. ACM.
- [112] A.R. Alameldeen, Z. Chishti, C. Wilkerson, Wei Wu, and Shih-Lien Lu. Adaptive cache design to enable reliable low-voltage operation. *IEEE Transactions on Computers*, 60(1):50–63, 2011.
- [113] T.N. Miller, R. Thomas, J. Dinan, B. Adcock, and R. Teodorescu. Parichute: Generalized turbocode-based error correction for near-threshold caches. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '10, pages 351–362, 2010.
- [114] Shi-Ting Zhou, S. Katariya, H. Ghasemi, S. Draper, and Nam Sung Kim. Minimizing total area of low-voltage sram arrays through joint optimization of cell size, redundancy, and ecc. In *Proceedings of the 2010 IEEE International Conference on Computer Design*, ICCD '10, pages 112–117, 2010.
- [115] Tayyeb Mahmood and Soontae Kim. Realizing near-true voltage scaling in variation-sensitive l1 caches via fault buffers. In *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '11, pages 85–94, New York, NY, USA, 2011. ACM.

- 
- [116] Young Geun Choi, Sungjoo Yoo, Sunggu Lee, and Jung Ho Ahn. Matching cache access behavior and bit error pattern for high performance low vcc l1 cache. In *Proceedings of the 48th Design Automation Conference, DAC '11*, pages 978–983, New York, NY, USA, 2011. ACM.
- [117] Abbas BanaiyanMofrad, Houman Homayoun, and Nikil Dutt. Fft-cache: a flexible fault-tolerant cache architecture for ultra low voltage operation. In *Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES '11*, pages 95–104, New York, NY, USA, 2011. ACM.
- [118] Alaa R. Alameldeen, Ilya Wagner, Zeshan Chishti, Wei Wu, Chris Wilkerson, and Shih-Lien Lu. Energy-efficient cache design using variable-strength error-correcting codes. In *Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA '11*, pages 461–472, New York, NY, USA, 2011. ACM.
- [119] Xuebei Yang and Kartik Mohanram. Unequal-error-protection codes in srams for mobile multimedia applications. In *Proceedings of the International Conference on Computer-Aided Design, ICCAD '11*, pages 21–27, Piscataway, NJ, USA, 2011. IEEE Press.
- [120] Gangyong Jia, Xi Li, Chao Wang, Xuehai Zhou, and Zongwei Zhu. Memory affinity: Balancing performance, power, thermal and fairness for multi-core systems. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing, CLUSTER '12*, pages 605–609, 2012.
- [121] Michael R. Jantz, Carl Strickland, Karthik Kumar, Martin Dimitrov, and Kshitij A. Doshi. A framework for application guidance in virtual memory systems. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '13*, pages 155–166, New York, NY, USA, 2013. ACM.
- [122] Kinam Kim and Jooyoung Lee. A new investigation of data retention time in truly nanoscaled drams. *IEEE Electron Device Letters*, 30(8):846–848, 2009.

- 
- [123] Nicholas Nethercote and Julian Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, pages 89–100, New York, NY, USA, 2007. ACM.
- [124] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Kathleen Baynes, Amer Jaleel, and Bruce Jacob. Dramsim: a memory system simulator. *ACM SIGARCH Computer Architecture News*, 33(4):100–107, November 2005.
- [125] Kevin Skadron, Mircea R. Stan, Karthik Sankaranarayanan, Wei Huang, Sivakumar Velusamy, and David Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1): 94–125, March 2004.
- [126] Micron Technology, Inc. 512mb: x4, x8, x16 ddr2 sdram features. <http://www.micron.com/~/media/Documents/Products/Data%20Sheet/DRAM/512MbDDR2.pdf>, 2004.
- [127] Standard Performance Evaluation Corporation. Spec cpu2000 and specjbb2005. <http://www.spec.org>, 2005.
- [128] John D. Davis, James Laudon, and Kunle Olukotun. Maximizing cmp throughput with mediocre cores. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, PACT '05, pages 51–62, Washington, DC, USA, 2005. IEEE Computer Society.
- [129] Jichuan Chang and Gurindar S. Sohi. Cooperative caching for chip multiprocessors. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 264–276, Washington, DC, USA, 2006. IEEE Computer Society.
- [130] Wangyuan Zhang and Tao Li. Exploring phase change memory and 3d die-stacking for power/thermal friendly, fast and durable memory architectures. In *Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*, PACT '09, pages 101–112, Washington, DC, USA, 2009. IEEE Computer Society.

- 
- [131] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2):305–327, 2003.
- [132] Micron Technology, Inc. 1gb: x4, x8, x16 ddr3 sdram features. http://www.micron.com/~/Documents/Products/Data%20Sheet/DRAM/1Gb_DDR3_SDRAM.pdf, 2006.
- [133] Barry W. Williams. Principles and elements of power electronics: Devices, drivers, applications, and passive components. *University of Strathclyde - Glasgow*, 2006.
- [134] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hällberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, February 2002.
- [135] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet’s general execution-driven multiprocessor simulator (gems) toolset. *ACM SIGARCH Computer Architecture News*, 33(4):92–99, November 2005.
- [136] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. Cacti 5.1. *HP Laboratories*, 2, 2008.
- [137] Wangyuan Zhang and Tao Li. Characterizing and mitigating the impact of process variations on phase change based memory systems. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO ’09*, pages 2–13, New York, NY, USA, 2009. ACM.
- [138] Aseem Agarwal, David Blaauw, Vladimir Zolotov, Savithiri Sundareswaran, Min Zhao, Kaushik Gala, and Rajendran Panda. Path-based statistical timing analysis considering inter-and intra-die correlations. In *Proceedings of the International Workshop on Timing Issues in the Specifications and Synthesis of Digital Systems, TAU ’02*, pages 16–21, 2002.

- 
- [139] Tezzaron Semiconductor. Soft errors in electronic memory - a white paper. http://www.tezzaron.com/media/soft_errors_1_1_secure.pdf, 2004.
- [140] S.S. Mukherjee, J. Emer, and S.K. Reinhardt. The soft error problem: an architectural perspective. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, HPCA '05, pages 243–247, 2005.